# 2021年北航计组P5课下设计报告

## 一 CPU设计方案简述

### （一）总体设计概述

本CPU为Verilog实现的流水线MIPS - CPU，支持的指令集包含{addu、subu、lw、sw、beq、lui、 ori 、jal、jr、nop}。为了实现这些功能，CPU主要包含了IFU、GRF、ALU、DM、NPC、EXT、CMP、CU、SU、流水线寄存器 D_Reg、E_Reg等模块。

### （二）关键模块设计

### 1. GRF

**GRF端口定义**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I | 时钟信号 |
| reset | I | 复位信号， 1 复位，0无效 |
| WE | I | 写使能信号，1写入，0不能写入 |
| A1 | I | 5位地址输入信号，将对应的寄存器数据读出到RD1 |
| A2 | I | 5位地址输入信号，将对应的寄存器数据读出到RD2 |
| A3 | I | 5位地址输入信号，将数WD存储到所对应的寄存器 |
| WD3 | I | 32位输入数据 |
| RD1 | O | 输出A1地址所对应的寄存器数据 |
| RD2 | O | 输出A2地址所对应的寄存器数据 |
| pc | I | 用于输出$display中的地址指令信息 |

**功能定义**

| 序号 | 功能名称 | 描述 |
| --- | --- | --- |
| 1 | 复位 | reset信号有效时，所有寄存器存储的数据清零 |
| 2 | 读数据(附带内部转发) | 读A1,A2所对应的寄存器数据到RD1，RD2，当A3=A1或A3=A2时，且写入信号为1时，直接将WD3端的数据输出到RD1或者RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿到来时，将WD3存入到A3所对应的寄存器，并且输出对应的地址指令以及相应的寄存器信息 |

内部转发代码

```
assign RD1 = (A1 == 0)? 32'b0:
             (A1 == A3 && en == 1'b1 )? WD:
                                        grf[A1];
assign RD2 = (A2 == 0)? 32'b0:
             (A2 == A3 && en == 1'b1 )? WD:
                                        grf[A2];
```

## 2. DM

**DM端口定义**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| A [31：0] | I | 读入五位地址信号 |
| WD [31:0] | I | 32位信号，存入A[11:2]所对应的地址 |
| clk | I | 时钟信号 |
| reset | I | 清零信号 |
| WE | I | 读入使能信号，当1时，存入数据 |
| RD [31:0] | O | 32位信号，输出A[11:2]所对应的寄存器的数据 |
| pc | I | 对应的指令地址信息 |

**DM功能描述**

| 序号 | 功能名称 | 描述 |
| --- | --- | --- |
| 1 | 清零 | 当reset信号位1时，里的数据清零 |
| 2 | 读入数据 | 当WE信号为1，将WD存储到addr对应的寄存器 |

**备注： DM的实质是3072*32bit的寄存器，addr为对应的寄存器地址，addr为12位，取A的13-2位。**

## 3.ALU

**ALU 端口简述**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| DataA [31:0] | I | 数据A |
| DataB [31:0] | I | 数据B |
| ALUctrl [2:0] | I | 运算控制选择信号 |
| result [31:0] | O | 输出结果 |
| ALUzero | O | 当result结果为0，ALUzero输出为1 |

**ALU 功能介绍**

| ALUctrl | 运算描述 |
| --- | --- |
| 000 | A&B |
| 001 | A\|B |
| 010 | A + B |
| 011 | A - B |
| 100 | A⊕B |

## 4. NPC

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| [31:0] D_pc | I | 位于D级的当前指令地址 |
| [31:0]F_pc | I | 位于F级的下一指令地址 |
| [25:0] imm | I | 指令立即数 |
| [31: 0] FW_rs_D | I | 当指令为jr时，经过转发后的需转跳的rs寄存器所存储的地址 |
| branch | I | beq指令信号 |
| cmp_result | I | beq指令判断信号 |
| jump | I | jump指令信号 |
| jr | I | jr指令判断信号 |
| [31:0] npc | O | 下一指令 |

**功能介绍**

1.jr信号为1时，npc为经过转发后的D级 rs寄存器的值。

2.jump信号为1时，npc的值为 D_pc31-28 || index || 00

3 .branch 和 cmp_result信号同时为1时，npc为 D_PC +4+ sign_ext(imm)||00

4.上述情况都不满足时，npc为Fpc+4

# 5.IFU

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I | 时钟信号 |
| reset | I | 清零信号 |
| [31:0] npc | I | 下一地址 |
| en | I | pc寄存器使能端 |
| F_pc | O | F所需取指的地址 |
| F_instr | O | F级所取出的指令 |

**功能介绍**

1. 根据复位信号，将F_pc地址信号同步复位至0x0000_3000
2. 根据当前npc的输入，在时钟上升沿，更新F_pc的输出
3. 取指，根据当前F_pc所指向的地址，取出指令
4. 冻结寄存器，使能端 en = ！ stall，当流水线需要暂停时，寄存器使能端为0，无法写入新的F_pc地址，寄存器被冻结。

## 6 . EXT

| 端口 | 方向 | 描述 |
| --- | --- | --- |
| [25:0] imm | I | 待扩展的立即数 |
| [1:0] EXT | I | 扩展控制信号 |
| D_imm | O | 扩展后的输出信号 |

控制实现代码

```
assign D_imm = ( EXT == 2'b00 )? {{16{imm[15]}},imm[15:0]}  :
               ( EXT == 2'b01 )? {{16{1'b0}},imm[15:0]}  :
               ( EXT == 2'b10 )? { imm[15:0],{16{1'b0}} } :
                                                    32'b0 ;
```

# （二）控制器CU设计思路

| 端口 | 方向 | 描述 |
| --- | --- | --- |
| [31:0] instr | I | 指令 |
| [4:0] rs_addr | O | rs寄存器地址 |
| [4:0] rt_addr | O | rt寄存器地址 |
| [4:0] rd_addr | O | rd寄存器地址 |
| [25:0] imm | O | 立即数 |
| [1:0] EXT | O | EXT模块控制信号 |
| branch | O | beq指令信号 |
| jump | O | jal、j指令信号 |
| jrtype | O | jr指令信号 |
| ALUSrc | O | ALU dataB端的数据选择信号 |
| RFWE | O | 寄存器写入信号 |
| DM_WE | O | DM写入信号 |
| [2:0] ALUctrl | O | ALU控制信号 |
| [1:0] RFWD_Type | O | 寄存器写入数据来源类型判断 |
| [4:0] RFaddr | O | 寄存器写入地址 |
| calc_r | O | 指令类型为 R信号 |
| calc_i | O | 指令类型为 I信号 |
| load | O | 指令类型为 load信号 |
| store | O | 指令类型为 store信号 |
| lui | O | 指令为 lui信号 |
| j_imm | O | 指令为直接转跳立即数的信号 |
| j_rs | O | 指令为转跳寄存器的存储地址的信号 |
| j_link | O | 指令为转跳并链接地址到寄存器的信号 |

**控制器CU主要实现3个功能：**

1.输出指令所对应的rs、rt、rd寄存器地址以及立即数。

2.输出指令所对应的各类模块的控制信号。

3.输出指令所属的指令类型，该功能主要用于SU模块，根据输出的指令类型更新Tuse和Tnew的值，用于分析是否需要暂停

**控制器对应真值表**

| 控制信号 | addu/subu | lw | sw | beq | j | ori |
|---|---|---|---|---|---|---|
| opcode | 000000 | 100011 | 101011 | 000100 | 000010 | 001101 |
| DM_WE | 0 | 0 | 1 | 0 | 0 | 0 |
| RFWE | 1 | 1 | 0 | 0 | 0 | 1 |
| ALUSrc | 0 | 1 | 1 | 0 | X | 1 |
| ALUctrl | 010/011 | 010 | 010 | 011 | XXX | 001 |
| branch | 0 | 0 | 0 | 1 | X | 0 |
| jump | 0 | 0 | 0 | 0 | 1 | 0 |
| EXT | 00 | 00 | 00 | 00 | XX | 01 |
| jr | 0 | 0 | 0 | 0 | 0 | 0 |

| 控制信号 | lui | jal | jr |
|---|---|---|---|
| opcode | 001111 | 000011 | 000000 |
| DM_WE | 0 | 0 | 0 |
| RFWE | 1 | 1 | 0 |
| ALUSrc | 1 | X | X |
| ALUctrl | 100 | XXX | XXX |
| branch | 0 | 0 | 0 |
| jump | 0 | 1 | 0 |
| EXT | 10 | XX | XX |
| jr | 0 | 0 | 1 |

控制器模块代码实现

```
wire [5:0] opcode;
   wire [5:0] func;
    assign opcode = instr [31:26];
    assign func = instr [5:0];

    assign rs_addr = instr [25:21];
    assign rt_addr = instr [20:16];
    assign rd_addr = instr [15:11];
    assign imm = instr [25:0];

    wire addu = ( opcode == 6'b000000 && func == 6'b100001 )? 1:0;
   wire subu = ( opcode == 6'b000000 && func == 6'b100011 )? 1:0;
    wire ori  = ( opcode == 6'b001101 )? 1:0;
    wire lw   = ( opcode == 6'b100011 ) ? 1:0;
    wire sw   = ( opcode == 6'b101011 )? 1:0;
```

```
    wire beq  = ( opcode == 6'b000100 )? 1:0;
    wire j    = ( opcode == 6'b000010 )? 1:0;
    wire jal  = ( opcode == 6'b000011 )? 1:0;
    wire jr   = ( opcode == 6'b000000 && func == 6'b001000 )? 1:0;
    assign  lui  = ( opcode == 6'b001111 )? 1:0;

    assign EXT = ( ori == 1'b1 )? 2'b01 :
                 ( lui == 1'b1 )? 2'b10 :
                                  2'b00;


   assign jump = j|jal;
   assign ALUSrc = lw | sw | ori | lui ;
    assign RFWE = addu | subu | lw | ori | lui | jal ;
    assign DM_WE = sw;
    assign jrtype = jr;

    assign ALUctrl = ( addu == 1'b1 || sw == 1'b1 || lw ==1'b1 )? 3'b010:
                     ( subu == 1'b1 ) ? 3'b011 :
                     ( ori == 1'b1 )? 3'b001 :
                     ( lui == 1'b1 )? 3'b100 :
                                      3'b000 ;

    assign calc_r = addu | subu;
    assign calc_i = ori;
    assign load = lw;
    assign store = sw;
    assign branch = beq;
    assign j_imm = j | jal;
    assign j_rs = jr;
    assign j_link = jal;

    assign RFaddr = calc_r ? instr[15:11]:
                    ( calc_i | lui | load)? instr[20:16]:
                          jal ? 5'd31 :
                                5'd0;
    assign RFWD_Type =  load ? 2'b01:
                           j_link ? 2'b10:
                                    2'b00 ;
```

# (三)暂停转发设计

## 1. 转发设置

### 确定转发位置

转发设置较为简单，首先是要判断哪些模块的输入接口需要转发，根据课件的知道，需要寄存器值输入的模块都需要转发，所以确定的所需转发地方如下：

1.ID级CMP的RS、RT寄存器输入端

2.ID级NPC的RS输入端同样也需要转发，但可以与CMP输入端共用一个转发选择器

3.EX级 ALU的dataA、dataB输入端

4.DM的数据输入端，当s执行sw指令时，需要使用寄存器的值

**确定转发来源**

当确定所需转发的端口后，我们在进行分析，转发数据的来源，显然，每一级转发的来源都来自其后各级流水线寄存器内的值。

ID级转发来源：E级流水线寄存器中的PC

M级流水线寄存器中的PC和ALUresult

W级流水线寄存器中的PC 、ALUresult以及DMresult

EX级转发来源：M级流水线寄存器中的PC和ALUresult

W级流水线寄存器中的PC 、ALUresult以及DMresult

DM级转发来源：W级流水线寄存器中的PC 、ALUresult以及DMresult

**判断是否需要转发**

当需求者需要使用的**寄存器地址**与转发来源的**写入寄存器地址**（即要求转发来源处**寄存器写入信号为1**） 相同且不为0时，转发。

**代码实现**

为了便于转发的选择器的书写以及结构清晰，我们可以将同一级的流水线寄存器中的输出做一个选择，只要在CU中加一个输入数据类型判断信号，根据输出类型信号选择转发的数据即可。比如，当前位于M级的指令为addu，类型为calc_R，所以M级的转发数据应该选择为 ALUresult。

选择示例如下：

```
assign W_WD = ( RFWDtype_W == 2'b00 )? W_ALUresult:
              ( RFWDtype_W == 2'b10 )? W_pc+8:
              ( RFWDtype_W == 2'b01 )? W_dm:
                                                  32'b0;
```

转发控制代码如下：

```
    wire [31:0] FW_rs_D = ( D_rs == 5'd0 )? 5'd0:
                          ( D_rs == RFaddr_E &&  RFWR_E == 1'b1 )?  E_WD:
                          ( D_rs == RFaddr_M &&  RFWR_M == 1'b1 )?  M_WD:
                          ( D_rs == RFaddr_W &&  RFWR_W == 1'b1 )?  W_WD:

RD1;
    wire [31:0] FW_rt_D = ( D_rt == 5'd0 )? 5'd0:
                          ( D_rt == RFaddr_E &&  RFWR_E == 1'b1 )?  E_WD:
                          ( D_rt == RFaddr_M &&  RFWR_M == 1'b1 )?  M_WD:
                          ( D_rt == RFaddr_W &&  RFWR_W == 1'b1 )?  W_WD:

RD2;
```

## 2.暂停设置

根据教程AT法的要求，**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 Tuse 小于对应 E 级或 M 级指令的 Tnew 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。

暂停控制信号以SU模块实现，在SU模块中，我们只需要根据位于D 、E、M级中的指令，计算D级的**Tuse_rs, Tuse_rt** 和 **Tnew_E, Tnew_M, Tnew_W。** 在实际效果中，由于Tnew_M一直为0或者不存在，所以不会触发暂停。

##### Tuse表

| 指令 | rs | rt |
| --- | --- | --- |
| addu | 1 | 1 |
| subu | 1 | 1 |
| ori | 1 | X |
| lw | 1 | X |
| sw | 1 | 2 |
| beq | 0 | 0 |
| lui | X | X |
| j | X | X |
| jal | X | X |
| jr | 0 | X |

**备注：由于转发机制是Tuse 小于对应 E 级或 M 级指令的 Tnew ，所以对于那些不使用寄存器的指令，我们需要将他们的Tuse X视为无穷大，代码中用一个比2大的数表示即可。**

**Tnew表**

| 指令 | E | M | W |
| --- | --- | --- | --- |
| addu | 1 | 0 | 0 |
| subu | 1 | 0 | 0 |
| ori | 1 | 0 | 0 |
| lw | 2 | 1 | 0 |
| sw | X | X | X |
| beq | X | X | X |
| lui | 1 | 0 | 0 |
| j | X | X | X |
| jal | 0 | 0 | 0 |
| jr | X | X | X |

**备注：由于转发机制是Tuse 小于对应 E 级或 M 级指令的 Tnew ，所以对于那些不产生寄存器新值的指令，我们需要将他们的Tnew X视为无穷小，代码中用一个小于等于0的数表示即可。**

## 暂停判断代码实现

```
module SU(
    input [31:0] D_instr,
    input [31:0] E_instr,
    input [31:0] M_instr,
    input [31:0] W_instr,
    output  stall
    );


     wire [4:0] D_rs_addr , D_rt_addr;
     wire D_branch , D_calc_r, D_calc_i, D_load, D_store, D_j_rs;
     wire [1:0] Tuse_rs , Tuse_rt;

     CU D_CU (
    .instr(D_instr),
    .rs_addr(D_rs_addr),
    .rt_addr(D_rt_addr),
    .branch(D_branch),
    .calc_r(D_calc_r),
    .calc_i(D_calc_i),
    .load(D_load),
    .store(D_store),
    .j_rs(D_j_rs)
    );

     assign Tuse_rs = ( D_calc_r | D_calc_i | D_load | D_store ) ? 2'b01:
                    ( D_branch | D_j_rs )? 2'b00:
                                                    2'b11;
    assign Tuse_rt =  D_store ? 2'b10 :
                      D_calc_r  ? 2'b01:
                      D_branch ? 2'b00 :
                                      2'b11;


    wire [4:0] E_A3;
   wire E_calc_r , E_calc_i , E_load, E_lui, E_WE;
    wire [1:0] Tnew_E;
    CU E_CU (
    .instr(E_instr),
    .RFWE(E_WE),
    .RFaddr(E_A3),
    .calc_r(E_calc_r),
    .calc_i(E_calc_i),
    .load(E_load),
    .lui(E_lui)
    );

     assign Tnew_E = E_load? 2'b10:
                    ( E_calc_r | E_calc_i | E_lui ) ? 2'b01:
                                                                2'b00;
    wire [4:0] M_A3;
   wire M_WE, M_load;

     CU  M_CU(
    .instr(M_instr),
```

```
    .RFWE(M_WE),
    .RFaddr(M_A3),
    .load(M_load)
    );

    assign Tnew_M = M_load ? 2'b01:
                                    2'b00;

   wire stall_rs_E ,stall_rs_M , stall_rt_E ,stall_rt_M;
    assign stall_rs_E = ( Tuse_rs < Tnew_E ) && ( D_rs_addr == E_A3 ) && ( E_A3
!= 5'd0) && (E_WE == 1'b1);
   assign stall_rs_M = ( Tuse_rs < Tnew_M ) && ( D_rs_addr == M_A3 ) && ( M_A3
!= 5'd0) && (M_WE == 1'b1);

    assign stall_rt_E = ( Tuse_rt < Tnew_E ) && ( D_rt_addr == E_A3 ) && ( E_A3
!= 5'd0) && (E_WE == 1'b1);
   assign stall_rt_M = ( Tuse_rt < Tnew_M ) && ( D_rt_addr == M_A3 ) && ( M_A3
!= 5'd0) && (M_WE == 1'b1);

    wire  stall_rs = stall_rs_E | stall_rs_M;
    wire  stall_rt = stall_rt_E | stall_rt_M;

    assign stall = stall_rs | stall_rt;

endmodule
```

# 二 测试数据

测试数据分为测P4的单周期基础指令，测需要转发的指令，测试需要暂停的指令。

转发指令示例：

```
ori $22, $0, 64
ori $1, $0, 116
ori $4, $0, 140
ori $28, $0, 228
ori $29, $0, 228
addu $18 $28 $22
addu $29 $1 $18
addu $4 $29 $18
addu $5 $4  $18
lw $20 100($0)
sw $24 100($0)
```

暂停指令示例：

```
.text
ori $t1, $t1, 125
ori $t2, $t2, 200
ori $t3, $t3, 25
ori $t4, $t4, 50
subu $t2, $t2, $t4
beq $t1, $t2, label1
sw $t1, 300($0)
```

```
sw $t2, 400($0)
sw $t3, 500($0)
label1:
lui $s1,  0x3000
lui $s2,  0x2000
subu $t2, $t2, $t3
beq $t1, $t2, label2
nop
sw $s1, 700($0)
sw $s2, 800($0)
label2:
sw $s1, 100($0)
sw $s2, 200($0)

ori $t1, $t1, 300
ori $t2, $t2, 100
sw $t2, 120($0)
beq  $t1, $t2, label3
nop
ori $s1, $s1,1
ori $s2, $s2,2
label3:
lw $t1, 120($0)
beq $t1,$t2, label4
nop
ori $s3, $s3,3
ori $s4, $s4,4
label4:
ori $5, $5, 5
ori $6, $6, 6

ori $t1, $t1, 400
ori $t2, $t2, 600
ori $t3, $t3, 100
ori $t4, $t4, 50
sw $t3, 100($0)
lw $t4, 100($0)
subu $t1, $t1,$t4
addu $t2, $t2, $t4
```

# 三 思考题

### 1.在采用本节所述的控制冒险处理方式下，PC 的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

在设计的流水线CPU中，因为有流水线的设计，所以无论当前指令是什么，下一指令必定执行。所以可以将计算下一指令地址pc的模块放在D级，此时下一指令已经进入F级的取指模块。D级根据当前指令计算下一指令地址，将计算出来的NPC写回到F级的PC的寄存器，在下一上升沿到来时通过寄存器写入将NPC输入给F级。

## 2.对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8？

因为由流水线延时槽的存在，jal指令的后一条指令是一定会在jal指令之后执行的，所以当转跳后需要jr转跳回时，转跳回的指令应该是jal指令后的第二条指令，所以需要回写PC+8.

## 3.为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由 ALU 或者 DM 等部件来提供数据?

我的理解是这样，如果是向D级转发，确实可以有E级的ALU和M级的DM来提供数据，但是如果是向E级和M级转发则不行。我们思考转发的实质是什么，以E级为例，转发的实质其实是我上一指令或者上两条指令的计算结果要作为现在的指令的源计算数，所以就需要一个寄存器把上一指令或者上两条指令的计算结果保存下来，这是转发的实质，所以也就说明了为什么我们转发的对象是流水级寄存器，而不是部件。

## 4.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。

不使用转发后的数据，会造成后续的指令源操作数引用错误。

代码示例

```
ori $t2,$t2,1
addu $t2, $s1, $s2
subu $t4, $t3, $t2
lw $t2,array($t3)
```

就比如这几条指令，subu指令在E级时需要转发 ￥￥￥￥￥￥

## 5.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?

```
addu $t1, $t2, $t3
subu $s1, $s2, $s3
subu $s3, $s4, $s5
addu $6, $t5, $1
```

寄存器的内部转发主要解决的问题，是比如给出示例中，第一条addu指令在对$t1寄存器进行写入操作时候，与第二条addu指令进行$t1寄存器的读取会发生冲突，无法保证读取的寄存器是写入的新值。如果不采用内部转发，可以将W级即将写入的数据转发是D级的CMP的rs、rt输入端。

## 6.为什么 0 号寄存器需要特殊处理?

因为0号寄存器始终不能写入值，默认值一直为0

## 7.什么是“最新产生的数据”?

最新产生的数据定义取决于哪一级的寄存器于当前指令所在的级最新，比如对于出于E级的指令，最新产生的数据位于M级的寄存器，如果M级的寄存器储存数据符合转发条件则转发，否则再考虑次新级的数据，即W级的转发数据。

## 8.在 AT 方法讨论转发条件的时候，只提到了“供给者需求者的 A 相同，且不为 0”，但在 CPU 写入 GRF 的时候，是有一个 we 信号来控制是否要写入的。为何在 AT 方法中不需要特判 we 呢？为了用且仅用 A 和 T 完成转发，在翻译出 A 的时候，要结合 we 做什么操作呢？

在翻译出A的时候，如果该指令不需要向寄存器写入值的话，则直接将A赋为0。

## 9.在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

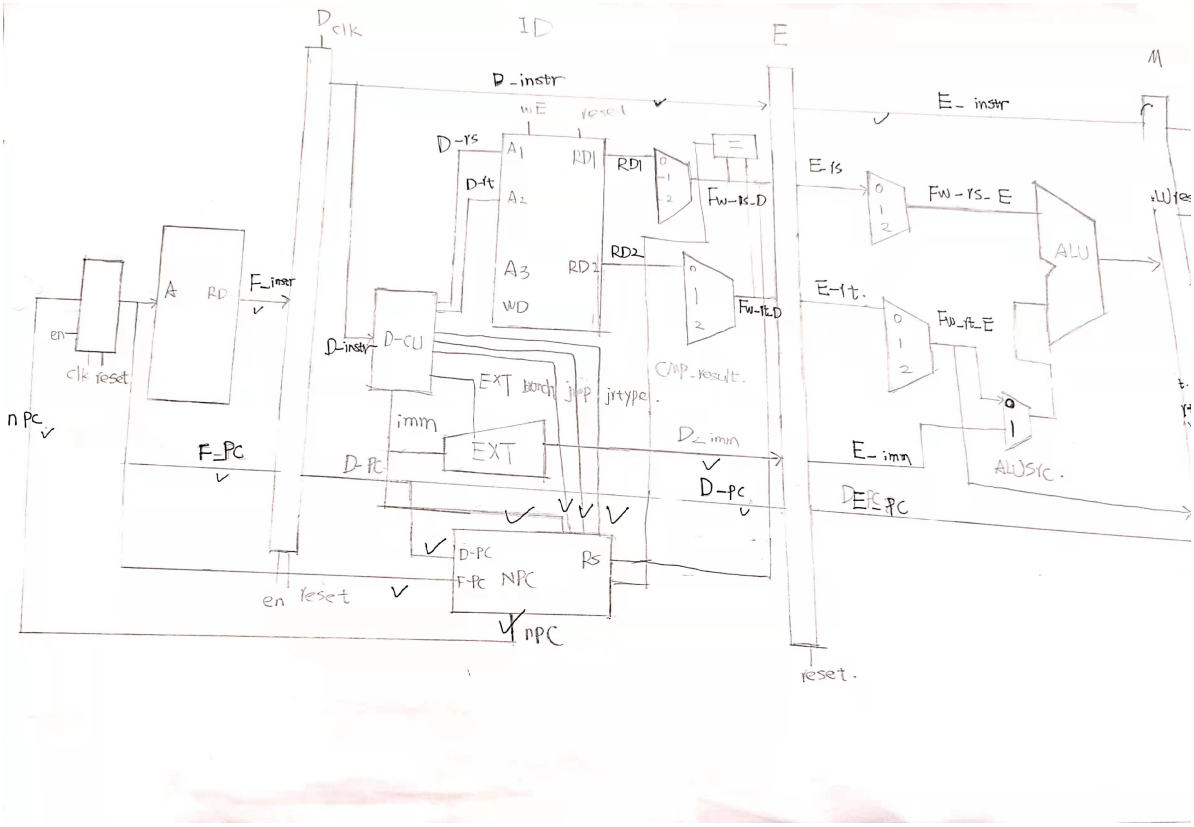
指令冲突见下表，解决方案通过上述的暂停机制解决冒险冲突，相应的测试样例见上测试代码。

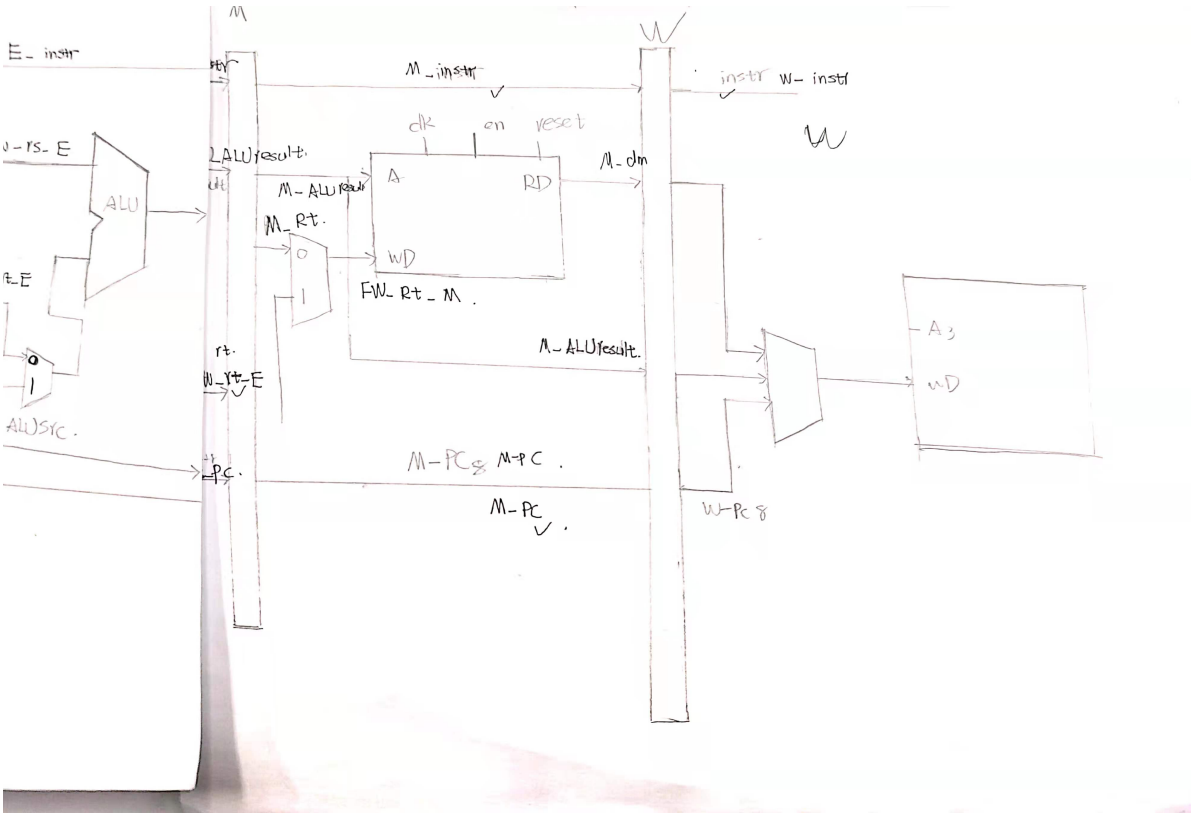
| | | addu | subu | ori | lw | sw | beq | lui | j | jal | jr |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Tnew_E | 1 | 1 | 1 | 2 | X | X | 1 | X | 0 | X |
| | Tuse_rs | | | | | | | | | | |
| addu | 1 | | | | | S | | | | | |
| subu | 1 | | | | | S | | | | | |
| ori | 1 | | | | | S | | | | | |
| lw | 1 | | | | | S | | | | | |
| sw | 1 | | | | | S | | | | | |
| beq | 0 | | S | S | S | S | | | S | | |
| lui | X | | | | | | | | | | |
| j | X | | | | | | | | | | |
| jal | X | | | | | | | | | | |
| jr | 0 | | S | S | S | S | | | S | | |

| | | addu | subu | ori | lw | sw | beq | lui | j | jal | jr |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Tnew_E | 1 | 1 | 1 | 2 | X | X | 1 | X | 0 | X |
| | Tuse_rt | | | | | | | | | | |
| addu | 1 | | | | | S | | | | | |
| subu | 1 | | | | | S | | | | | |
| ori | X | | | | | | | | | | |
| lw | X | | | | | | | | | | |
| sw | 2 | | | | | | | | | | |
| beq | 0 | | S | S | S | S | | | S | | |
| lui | X | | | | | | | | | | |
| j | X | | | | | | | | | | |
| jal | X | | | | | | | | | | |
| jr | X | | | | | | | | | | |

| | | addu | subu | ori | lw | sw | beq | lui | j | jal | jr |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Tnew_M | 0 | 0 | 0 | 1 | X | X | 0 | X | 0 | X |
| | Tuse_rt | | | | | | | | | | |
| addu | 1 | | | | | | | | | | |
| subu | 1 | | | | | | | | | | |
| ori | X | | | | | | | | | | |
| lw | X | | | | | | | | | | |
| sw | 2 | | | | | | | | | | |
| beq | 0 | | | | | S | | | | | |
| lui | X | | | | | | | | | | |
| j | X | | | | | | | | | | |
| jal | X | | | | | | | | | | |
| jr | X | | | | | | | | | | |

| | | addu | subu | ori | lw | sw | beq | lui | j | jal | jr |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Tnew_M | 0 | 0 | 0 | 1 | X | X | 0 | X | 0 | X |
| | Tuse_rs | | | | | | | | | | |
| addu | 1 | | | | | | | | | | |
| subu | 1 | | | | | | | | | | |
| ori | 1 | | | | | | | | | | |
| lw | 1 | | | | | | | | | | |
| sw | 1 | | | | | | | | | | |
| beq | 0 | | | | | S | | | | | |
| lui | X | | | | | | | | | | |
| j | X | | | | | | | | | | |
| jal | X | | | | | | | | | | |
| jr | 0 | | | | | S | | | | | |


D_clk
ID
E
M
D_instr
E_instr
wE
reset
D-rs
D-rt
A1
A2
A3
WD
RD1
RD2
=
Fw-rs-D
Fw-rt-D
E-rs
E-rt.
Fw-rs-E
Fw-rt-E
ALU
F_instr
A
RD
en
clk
reset
D_instr
D-CU
EXT
branch
jump
jr+type.
CMP_result.
imm
EXT
D_imm
E_imm
ALUSrc.
nPC
F_PC
D-PC
E_PC
D-PC
F-PC
NPC
Rs
en
reset
nPC
reset.


M
W
E_instr
M_instr
instr
W_instr
clk
en
reset
ALU
M_ALUresult.
M_dm
A
RD
WD
M_Rt.
FW_Rt_M.
rt.
ALUSrc.
M_PC & M_PC.
M_PC.
W_PC &
A3
WD

# P5上机要点

## 转跳指令：branch +link指令

1.一些指令题目要求不满足判断条件，需要清空延时槽

- 如果需要清空延迟槽，将branch信号传给control_1,如果branch && instr_1为1，则将reset_01置1传给IF_module
- 注意当处于stall时，不reset，即reg_01内为 if(reset||(reset_01 && en_01))

2.是否写入条件判断RF

D级CMP的判断信号需要流水线一级一级传下去，根据指令信号和判断信号更新写入条件

也可以根据指令信号和判断信号，更新写入寄存器的地址，符合条件为31，不符合条件为0

D_branch _link 信号控制 Tuse_rt  Tuse_rs,

branch_link 在CU 控制 RFaddr 、RFWD_Type、btype、RFWE

在NPC里控制 npc