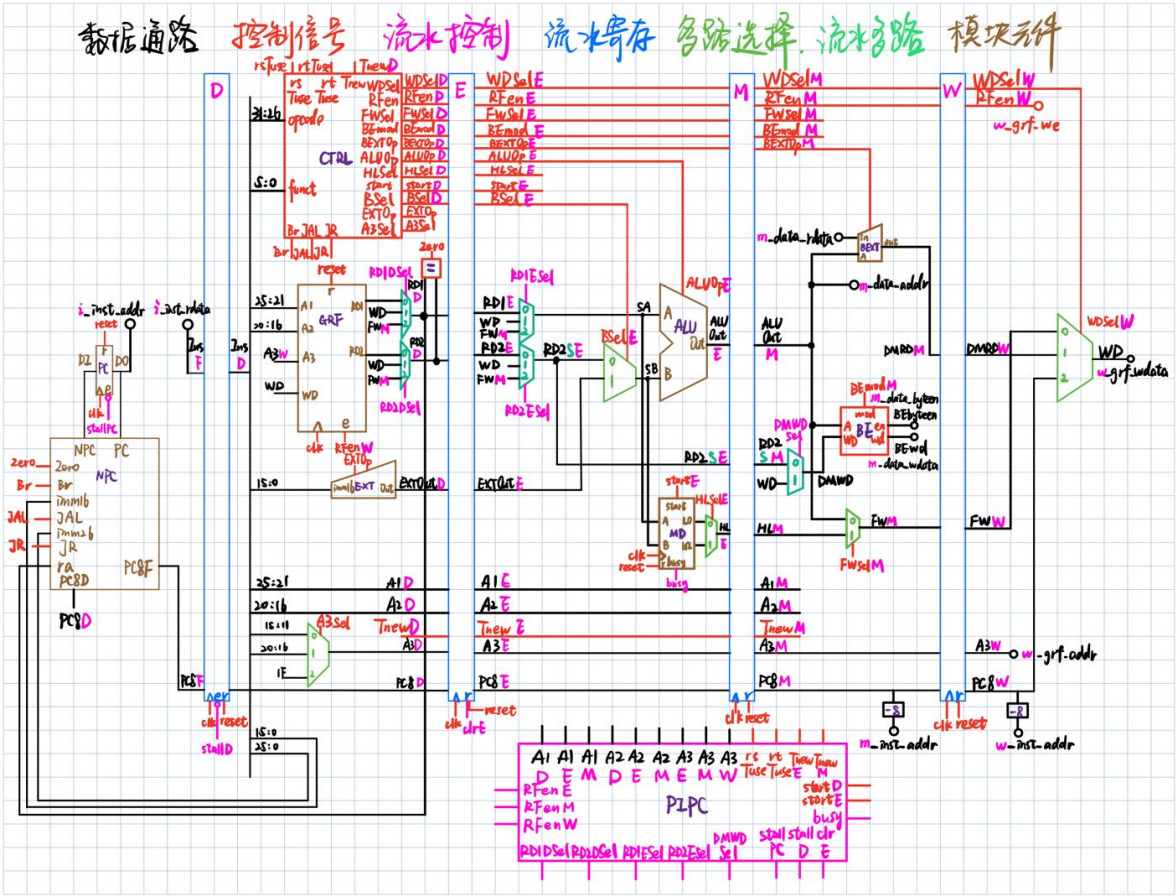
P7 MIPS微系统设计

一、设计草稿

1.（P6）CPU电路图与模块建模



2.主控制器指令真值表

指令	opcode	funct	Br	JAL	JR	A3Sel	EXTOp	BSel	HLSel	start	ALUOp	BEXTOp	BEmod	FWSel	RFen	WDSel	rsTuse	rtTuse	Tnew	指令
add	000000	100000	00	0	0	00	xx	0	x	0000	0000	xxx	00	0	1	00	1	1	2	add
sub	000000	100010	00	0	0	00	xx	0	x	0000	0001	xxx	00	0	1	00	1	1	2	sub
and	000000	100100	00	0	0	00	xx	0	x	0000	0010	xxx	00	0	1	00	1	1	2	and
or	000000	100101	00	0	0	00	xx	0	x	0000	0011	xxx	00	0	1	00	1	1	2	or
slt	000000	101010	00	0	0	00	xx	0	x	0000	0100	xxx	00	0	1	00	1	1	2	slt
sltu	000000	101011	00	0	0	00	xx	0	x	0000	0101	xxx	00	0	1	00	1	1	2	sltu
addi	001000	xxxxxx	00	0	0	01	01	1	x	0000	0000	xxx	00	0	1	00	1	x(5)	2	addi
andi	001100	xxxxxx	00	0	0	01	00	1	x	0000	0010	xxx	00	0	1	00	1	x(5)	2	andi
ori	001101	xxxxxx	00	0	0	01	00	1	x	0000	0011	xxx	00	0	1	00	1	x(5)	2	ori
lui	001111	xxxxxx	00	0	0	01	10	1	x	0000	0110	xxx	00	0	1	00	1	x(5)	2	lui
lw	100011	xxxxxx	00	0	0	01	01	1	x	0000	0000	000	00	x	1	01	1	x(5)	3	lw
lh	100001	xxxxxx	00	0	0	01	01	1	x	0000	0000	001	00	x	1	01	1	x(5)	3	lh
lb	100000	xxxxxx	00	0	0	01	01	1	x	0000	0000	010	00	x	1	01	1	x(5)	3	lb
sw	101011	xxxxxx	00	0	0	xx	01	1	x	0000	0000	xxx	01	x	0	xx	1	2	0	sw
sh	101001	xxxxxx	00	0	0	xx	01	1	x	0000	0000	xxx	10	x	0	xx	1	2	0	sh
sb	101000	xxxxxx	00	0	0	xx	01	1	x	0000	0000	xxx	11	x	0	xx	1	2	0	sb
mult	000000	011000	00	0	0	xx	xx	0	x	0001	xxxx	xxx	00	x	0	xx	1	1	0	mult
multu	000000	011001	00	0	0	xx	xx	0	x	0010	xxxx	xxx	00	x	0	xx	1	1	0	multu
div	000000	011010	00	0	0	xx	xx	0	x	0011	xxxx	xxx	00	x	0	xx	1	1	0	div
divu	000000	011011	00	0	0	xx	xx	0	x	0100	xxxx	xxx	00	x	0	xx	1	1	0	divu
mfhi	000000	010000	00	0	0	00	xx	x	1	0101	xxxx	xxx	00	1	1	00	x(5)	x(5)	2	mfhi
mflo	000000	010010	00	0	0	00	xx	x	0	0110	xxxx	xxx	00	1	1	00	x(5)	x(5)	2	mflo
mthi	000000	010001	00	0	0	xx	xx	x	x	0111	xxxx	xxx	00	x	0	xx	1	x(5)	0	mthi
mtlo	000000	010011	00	0	0	xx	xx	x	x	1000	xxxx	xxx	00	x	0	xx	1	x(5)	0	mtlo
nop	000000	000000	00	0	0	xx	xx	0	x	0000	xxxx	xxx	00	x	0	xx	x(5)	x(5)	0	nop
beq	000100	xxxxxx	01	0	0	xx	xx	0	x	0000	xxxx	xxx	00	x	0	xx	0	0	0	beq
bne	000101	xxxxxx	10	0	0	xx	xx	0	x	0000	xxxx	xxx	00	x	0	xx	0	0	0	bne
jal	000011	xxxxxx	00	1	0	10	xx	x	x	0000	xxxx	xxx	00	x	1	10	x(5)	x(5)	3	jal
jr	000000	001000	00	0	1	xx	xx	x	x	0000	xxxx	xxx	00	x	0	xx	0	x(5)	0	jr
指令	opcode	funct	Br	JAL	JR	A3Sel	EXTOp	BSel	HLSel	start	ALUOp	BEXTOp	BEmod	FWSel	RFen	WDSel	rsTuse	rtTuse	Tnew	指令

Tuse：数据用不用？用到哪里？

Tnew：数据产生不产生？还要多久产生？

Tuse<Tnew：暂停

Tuse>=Tnew：转发

3.P6指令涉及模块与指令分类

a.考虑不同类别指令涉及的不同模块，指令分类有：

（1）算数运算：add、sub、and、or、slt、sltu

涉及IFUF、GRFD、ALU、GRFW

（2）立即数运算：addi、andi、ori、lui

涉及IFUF、GRFD、EXT、ALU、GRFW

（3）内存访问：lw、lh、lb

涉及IFUF、GRFD、EXT、ALU、BEXT、GRFW

（4）内存写入：sw、sh、sb

涉及IFUF、GRFD、EXT、ALU、BE

（5）分支、返回：beq、bne、jr

涉及IFUF、GRFD、IFUD

（6）跳转：jal

涉及IFUF、IFUD、GRFW

（7）乘除运算：mult、multu、div、divu

涉及IFUF、GRFD、MD

（8）乘除访问：mfhi、mflo

涉及IFUF、MD、GRFW

（9）乘除写入：mthi、mtlo

涉及IFUF、GRFD、MD

b.针对每条指令涉及的特定模块，考虑信息有：

（1）模块内部实现

（2）模块接口需求

（3）模块接口来源（是否需要流水控制）

（4）控制信号设置（主控制+冒险控制、基础译码+新增扩展）

（5）流水传递寄存

4.改动设计

（1）CPU-CP0

```
module CP0(
    input clk,
    input reset,
    input en,
    input [4:0] CP0Add,
    input [31:0] CP0In,
    output [31:0] CP0Out,
    input [31:0] VPC,
    input BDIn,
    input [4:0] ExcCodeIn,
    input [5:0] HWInt,
    input EXLClr,
    output [31:0] EPCOut,
    output Req
    );
    reg [31:0] R [12:14];
    wire [31:2] EPC;
    initial begin
        R[12] <= 32'd0;
        R[13] <= 32'd0;
        R[14] <= 32'd0;
    end
    always@(posedge clk)begin
        if(reset)begin
            R[12] <= 32'd0;
            R[13] <= 32'd0;
            R[14] <= 32'd0;
        end else begin
            if(en &&(CP0Add >= 12)&&(CP0Add <= 14))begin
                R[CP0Add] <= CP0In;
            end
            if(Req)begin
                R[13][6:2] <= ExcCodeIn;
                R[13][31] <= BDIn;
```

```
                R[12][1] <= 1'b1;
                R[14][31:2] <= EPCOut[31:2];
            end else if(EXLClr)begin
                R[12][1] <= 1'b0;
            end
            R[13][15:10] <= HWInt;
        end
    end
    assign Req = ((|(HWInt & R[12][15:10]))&&(~R[12][1])&&(R[12][0]))||
((|ExcCodeIn)&&(~R[12][1]));
    assign EPC = (BDIn) ? (VPC[31:2] - 32'd1) : VPC[31:2];
    assign EPCOut = (Req) ? {EPC,2'b00} : {R[14][31:2],2'b00};
    assign CP0Out = ((CP0Add >= 12)&&(CP0Add <= 14)) ? R[CP0Add] : 32'd0;
endmodule
```

(2) Bridge

```
module Bridge(
    input [31:0] A,
    input [3:0] PrByteen,
    input [31:0] PrWD,
    output [31:0] PrRD,
    output [31:0] DMAddr,
    output [31:0] TC0Addr,
    output [31:0] TC1Addr,
    output [31:0] IGAddr,
    output [3:0] DMByteen,
    output TC0WE,
    output TC1WE,
    output [3:0] IGByteen,
    output [31:0] DevWD,
    input [31:0] DMRD,
    input [31:0] TC0RD,
    input [31:0] TC1RD
    );
    wire hitDM;
    wire hitTC0;
    wire hitTC1;
    wire hitIG;
    assign hitDM = (A >= 32'h0000)&&(A <= 32'h2fff);
    assign hitTC0 = (A >= 32'h7f00)&&(A <= 32'h7f0b);
    assign hitTC1 = (A >= 32'h7f10)&&(A <= 32'h7f1b);
    assign hitIG = (A >= 32'h7f20)&&(A <= 32'h7f23);
    assign PrRD = (hitDM) ? DMRD :
                          (hitTC0) ? TC0RD :
                          (hitTC1) ? TC1RD :
                          32'd0;
    assign DMByteen = (hitDM) ? PrByteen : 4'd0;
    assign TC0WE = hitTC0 &&(|PrByteen);
    assign TC1WE = hitTC1 &&(|PrByteen);
    assign IGByteen = (hitIG) ? PrByteen : 4'd0;
    assign DevWD = PrWD;
    assign DMAddr = A;
    assign TC0Addr = A;
    assign TC1Addr = A;
    assign IGAddr = A;
endmodule
```

(3) mips

```
//Exception F
    wire [4:0] ExcF;
    wire [31:0] InsF_Exc;
    assign ExcF = ((i_inst_addr % 4 != 0)||(i_inst_addr < 32'h3000)||
(i_inst_addr > 32'h6ffc)) ? 5'd4 : 5'd0;
    assign InsF_Exc = (ExcF != 5'd0) ?  32'd0 : InsF;
//Exception D
    wire [4:0] ExcD;
    assign ExcD = (ExcFD != 5'd0) ? ExcFD :
                        ExcCtrl;
    wire [4:0] ACP0D;
    assign ACP0D = (MFC0D || MTC0D) ? InsD[15:11] : 5'd0;
//Exception E
    wire [4:0] ExcE;
    assign ExcE = (ExcDE != 5'd0) ? ExcDE :
                        ExcALU;
//Exception M
    assign macroscopic_pc = m_inst_addr;
    wire [4:0] ExcM;
    assign ExcM = (ExcEM != 5'd0) ? ExcEM :
                        (ExcBE != 5'd0) ? ExcBE :
                        ExcBEXT;
    wire [4:0] ExcCodeIn;
    wire [31:0] CP0OutM;
    wire TCI0;
    wire TCI1;
    wire [31:0] VPC;
    assign ExcCodeIn = (interrupt == 1'b1 || TCI1 == 1'b1 || TCI0 == 1'b1) ?
5'd0 : ExcM;
    assign HWInt = {3'b000,interrupt,TCI1,TCI0};
    assign VPC = PC8M - 32'd8;
```

二、测试程序

```
# 程序首先从这里运行
.text
    # 只允许外部中断
    ori $t0, $0, 0x1001
    mtc0 $t0, $12

    # 算术溢出
    lui $t0, 0x7fff
    lui $t1, 0x7fff
    add $t2, $t0, $t1

end:
    beq $0, $0, end
    nop

.ktext 0x4180
_entry:
    # 保存上下文
    la $s0,_save_context
```

```
    jr $s0
    nop

_main_handler:
    # 取出 ExcCode
    mfc0 $k0, $13
    ori $k1, $0, 0x7c
    and $k0, $k0, $k1

    # 如果是中断，直接恢复上下文
    beq $k0, $0, _restore_context
    nop

    # 将 EPC + 4，即处理异常的方法就是跳过当前指令
    mfc0 $k0, $14
    add $k0, $k0, 4
    mtc0 $k0, $14
    la $s1,_restore_context
    jr $s1
    nop

_exception_return:
    eret

_save_context:
    ori $k0, $0, 0x1000     # 在栈上找一块空间保存现场
    addi $k0, $k0, -256
    sw $sp, 116($k0)        # 最先保存栈指针
    add $sp,$0, $k0

    # 依次保存通用寄存器（注意要跳过 $sp）、HI 和 LO
    sw $1, 4($sp)
    sw $2, 8($sp)
    # ......
    sw $31, 124($sp)
    mfhi $k0
    mflo $k1
    sw $k0, 128($sp)
    sw $k1, 132($sp)

    la $s2, _main_handler
    jr $s2
    nop

_restore_context:
    # 依次恢复通用寄存器（注意要跳过 $sp）、 HI 和 LO
    lw $1, 4($sp)
    lw $2, 8($sp)
    # ......
    lw $31, 124($sp)
    lw $k0, 128($sp)
    lw $k1, 132($sp)
    mthi $k0
    mtlo $k1

    # 最后恢复栈指针
    lw $sp, 116($sp)
```

```
    la $s3,_exception_return
    jr $s3
    nop
```

三、思考题

1、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?

通过南桥将位置位移及点击信息传送给cpu。

2、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）

保证中断异常处理程序不会和主程序冲突，保证不同环境的兼容，保证在遇到中断异常时能够完整地执行处理程序。

可以。但可能会出现嵌套异常等无法处理的情况。

3、为何与外设通信需要 Bridge?

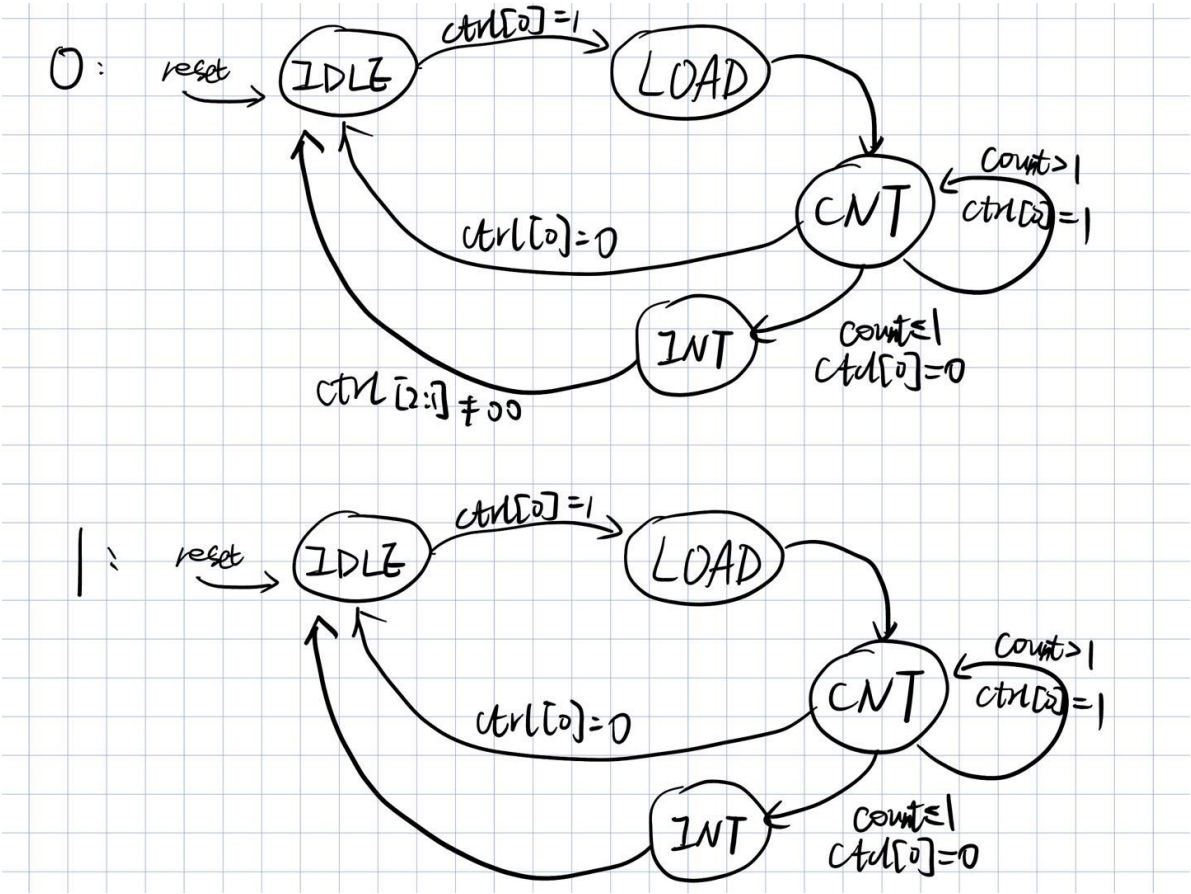
避免不同设备间的访问冲突。使其满足“高内聚低耦合”的封装思想，让操作更加简洁。

4、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，并分别针对每一种模式绘制状态移图。

模式0：产生定时中断。

模式1：产生周期性脉冲。

状态转移图：



5、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？

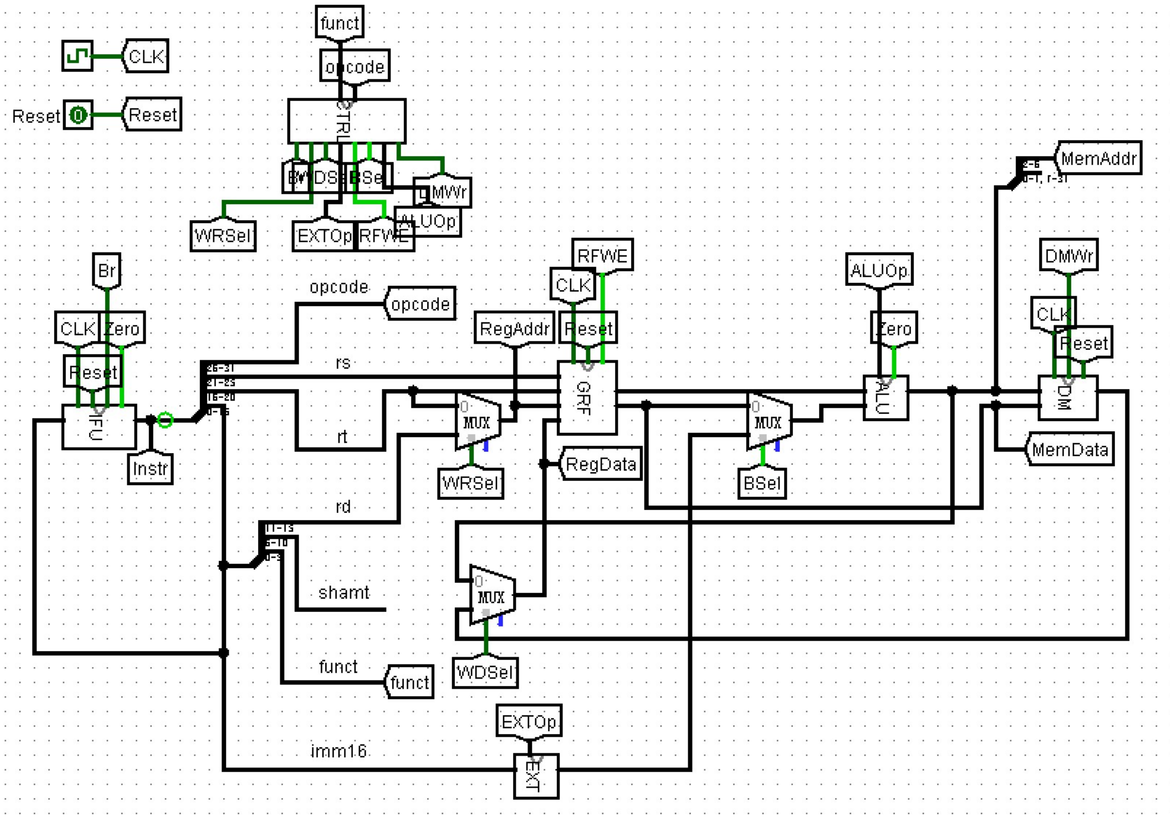
可能会导致返回时错误。

PC、BDIn。

6、为什么 `jalr` 指令为什么不能写成 `jalr $31, $31`？

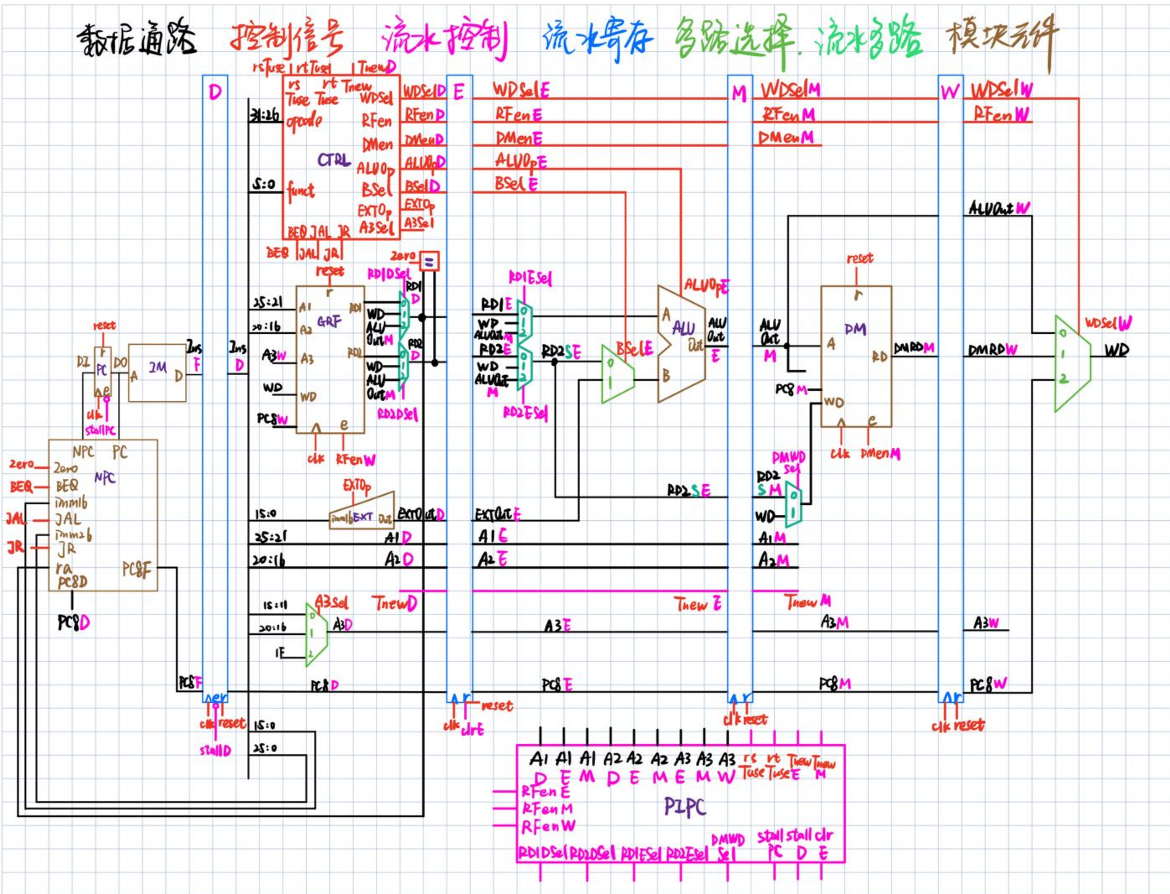
若需要在W级接受转发信号，则会产生31号寄存器的冲突。

附：P3电路图



CLK
Reset
funct
opcode
CTRL
WRSel
EXTOp
RFWE
MWr
ALUOp
Br
Zero
IFU
Instr
rs
rt
rd
shamt
funct
imm16
MUX
RegAddr
GRF
RegData
BSel
ALU
MemAddr
DMWr
DM
MemData
WDSel
EXT

附：P5电路图



数据通路
控制信号
流水控制
流水寄存
多路选择、流水旁路
模块元件
CTRL
GRF
ALU
DM
EXT
NPC
IM
PIPC