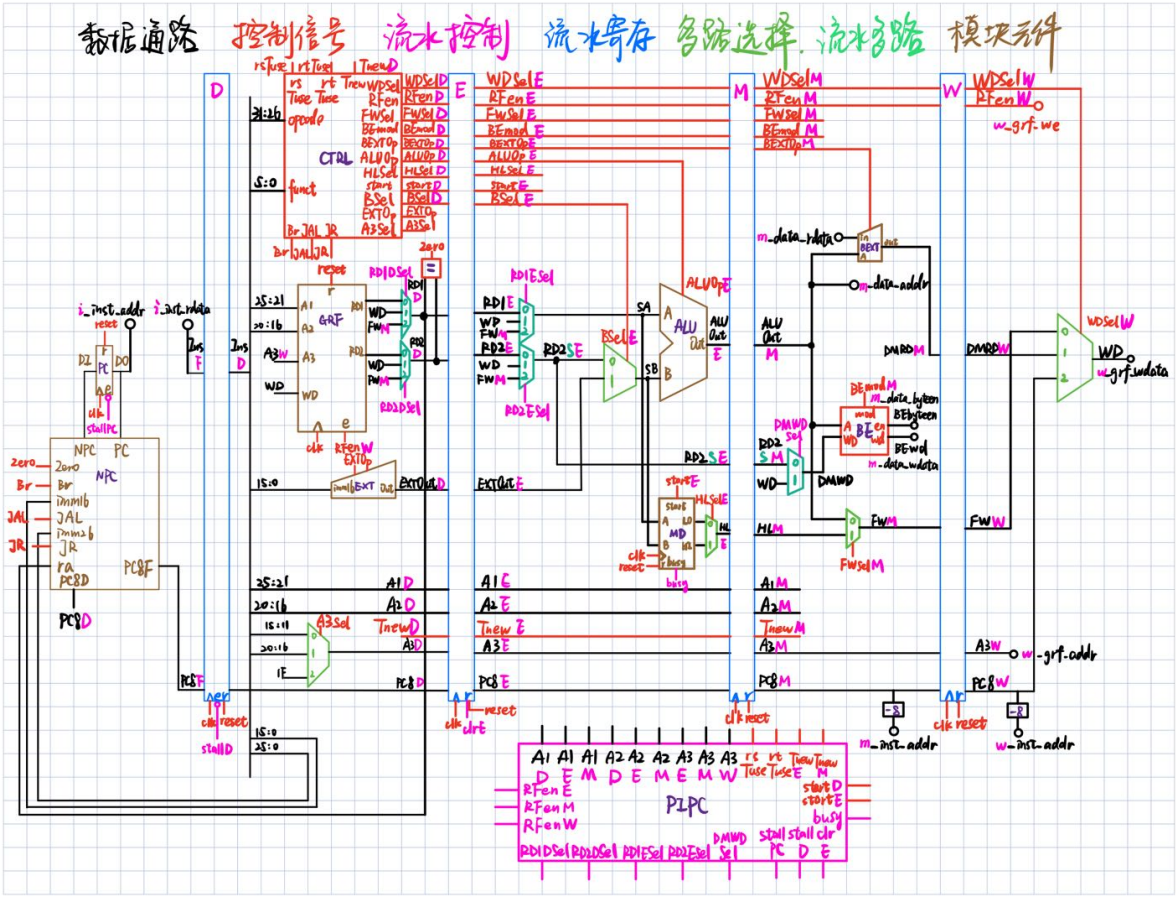
P6 流水线CPU设计

一、设计草稿

1.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.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

二、测试程序

1.ALU计算

```
.text
ori $1,$0,1
ori $2,$0,2
ori $3,$0,3
ori $4,$0,4
lui $8,0xffff
ori $8,$8,0xffff
addi $5,$1,1
andi $6,$3,2
slt $7,$2,$1
sltu $9,$2,$8
```

2.访存

```
.text
lui $1,0x1234
ori $1,$1,0x5678
sw $1,0($0)
sh $1,6($0)
sb $1,4($0)
lb $2,4($0)
lh $3,6($0)
lw $4,0($0)
```

3.跳转

```
.text
ori $3,2
label:
addi $1,$1, 1
mtlo $1
mflo $2
bne $3, $2, label
nop
```

4.乘除计算

```
.text
ori $1,$0,1
ori $2,$0,2
ori $3,$0,3
ori $4,$0,4
lui $5,0xffff
ori $5,$5,0xffff
mult $1,$5
nop
mfhi $6
nop
mflo $7
nop
multu $1,$5
nop
mfhi $6
nop
mflo $7
nop
div $1,$5
nop
mfhi $6
nop
mflo $7
nop
divu $1,$5
nop
mfhi $6
nop
mflo $7
nop
mthi $2
nop
mfhi $6
nop
mtlo $3
nop
mflo $7
nop
```

5.转发

```
.text
ori $1,$0,1
ori $2,$0,4
mflo $8
mfhi $9
ori $6,$0,6
mthi $6
divu $6,$1
beq $3,$6,C
divu $6,$2
sh $2,0($0)
mflo $8
mfhi $9
mult $2,$6
mflo $8
mfhi $9
lh $3,0($0)
multu $6,$1
mfhi $7
mflo $7
lb $7,1($2)
mtlo $7
mflo $7
addi $4,$3,1
slt $1,$3,$4
mult $1,$2
jal A
div $6,$2
lw $5,0($0)
mfhi $ra
mflo $ra
ori $ra,$0,0x3008
B:
add $3,$3,$1
A:
mflo $8
mfhi $9
beq $5,$3,B
mflo $6
sb $6,5($0)
mult $1,$2
jr $ra
mtlo $0
C:
mflo $8
mfhi $9
```

6.debug

```
.text
lui $8,0x0102
ori $8,$8,0x0304
lui $9,0x0001
ori $9,$9,0x0002
```

```
ori $10,$10,1
sw $8,0($0)
sw $8,4($0)
sw $9,8($0)
sw $10,12($0)
lb $1,1($0)
lb $2,1($1)
lh $3,4($2)
lh $4,4($3)
```

三、思考题

1、为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

由于乘除法需要多个周期才能完成，若整合进ALU将降低CPU效率，并且乘除法不需要写入GRF和DM。

由于设置了单独的乘除法部件，因此为了防止阻塞后无法区分存入寄存器指令就必须设置独立的HI、LO寄存器。

2、真实的流水线 CPU 是如何使用实现乘除法的？请查阅相关资料进行简单说明。

乘法：通过移位和累加完成。

除法：通过移位和累减完成。

3、请结合自己的实现分析，你是如何处理 Busy 信号带来的周期阻塞的？

```
//乘除模块
    always@(posedge clk)begin
        if(reset)begin
            count <= 4'd0;
            HIreg <= 32'd0;
            LOreg <= 32'd0;
        end else if((|start)== 1'b1)begin
            case(start)
                4'b0001:begin
                    count <= 4'd5;
                    HIreg <= mult[63:32];
                    LOreg <= mult[31:0];
                end
                4'b0010:begin
                    count <= 4'd5;
                    HIreg <= multu[63:32];
                    LOreg <= multu[31:0];
                end
                4'b0011:begin
                    count <= 4'd10;
                    HIreg <= mod;
                    LOreg <= div;
                end
                4'b0100:begin
                    count <= 4'd10;
```

```
                    HIreg <= modu;
                    LOreg <= divu;
                end
                4'b0101:begin
                    //nothing
                end
                4'b0110:begin
                    //nothing
                end
                4'b0111:begin
                    HIreg <= SA;
                end
                4'b1000:begin
                    LOreg <= SA;
                end
            endcase
        end else begin
            if(count > 4'd0)begin
                count <= count - 1;
            end
        end
    end
    assign busy = (count > 4'd0) ? 1'b1 : 1'b0;
//冒险控制器
    assign EisMD = (startE == 4'b0001)||(startE == 4'b0010)||
                   (startE == 4'b0011)||(startE == 4'b0100);
    assign stallMD = ((EisMD)|| busy)&&((|startD));
    assign STALL = stallrsE || stallrsM || stallrtE || stallrtM || stallMD;
```

通过不同运算来为计数器count赋值，随后每个周期递减1直至0，若count大于0则busy为1。

若E级指令为乘除运算或正在进行运算（busy==1），且D级指令涉及乘除模块则阻塞。

4、请问采用字节使能信号的方式处理写指令有什么好处？（提示：从清晰性、统一性等角度考虑）

清晰性：使用4位byteen信号即可清晰展现访存的字节位置。

统一性：使用统一的byteen信号统一了不同访存指令的写入情况。

5、请思考，我们在按字节读和按字节写时，实际从 DM 获得的数据和向 DM 写入的数据是否是一字节？在什么情况下我们按字节读和按字节写的效率会高于按字读和按字写呢？

不是。

按字节编址是按字节访存效率高。

6、为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

1.我将byteen信号产生过程和store指令数据调整过程整体封装在BE模块中，这样BEmod信号与A信号可以只用使用一次。

2.我将start信号和MDOp信号结合，通过`|start`来代替原有start一位信号，用`start`代替MDOp，在减少了译码复杂度和接口数量。

3.我将M级转发的ALUOut和HL信号通过FWSel选择为FWM后，将其流水到W级，在WDSel选择时减少了MUX端口数。

7、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？

乘除运算指令与乘除访问指令的冲突。

通过上述改进start信号判断阻塞解决。

测试样例：（见：二、4.）。

8、如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。

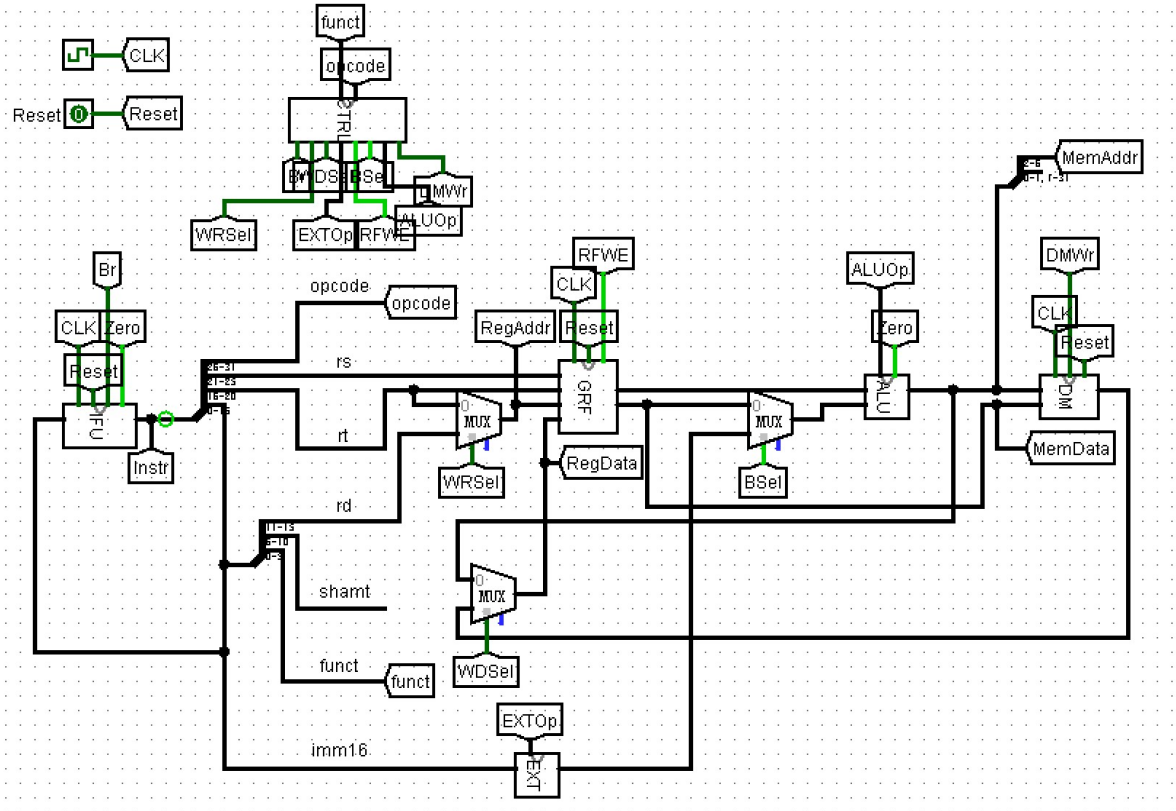
手动构造样例。

1.通过ALU计算、访存跳转、乘除计算、转发等方向入手进行构造。

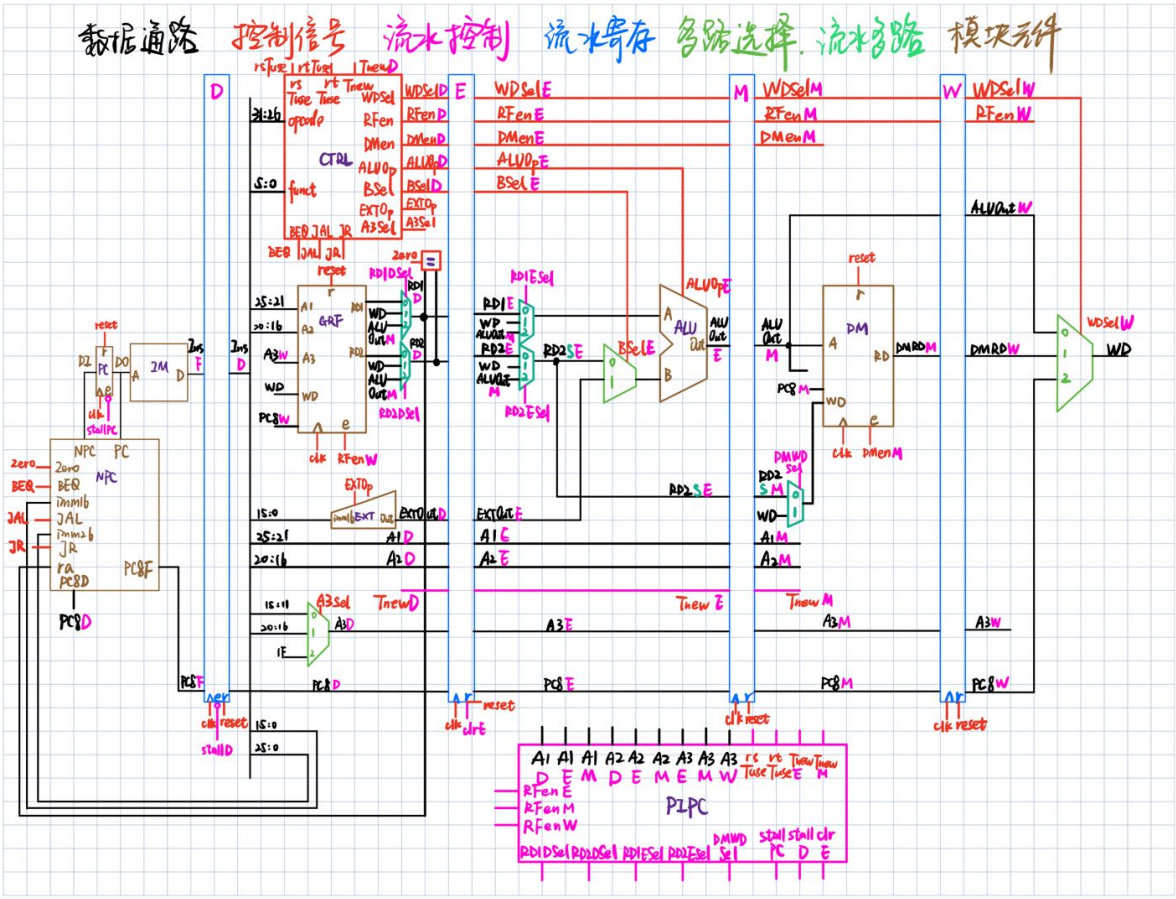
2.先结合改动之处可能产生的与其他指令的冲突进行枚举（乘除模块和字节访存模块）。

3.后结合流水线Tuse、Tnew的极端数据情况进行特别测试。

附：P3电路图



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 流水多路 模块元件
D
E
M
W
NPC
PC
zero
BEQ
JAL
JR
PC8F
PC8D
GRF
CTRL
opcode
funct
ALUOp
BSel
EXTOp
A3Sel
RFenW
ALU
DM
reset
clk
stallPC
stallD
clrE
PC8E
PC8M
PC8W
A3E
A3M
A3W
TnewD
TnewE
TnewM
WDSelD
WDSelE
WDSelM
WDSelW
RFenD
RFenE
RFenM
RFenW
DMenD
DMenE
DMenM
ALUOpD
ALUOpE
BSelD
BSelE
EXTOpD
EXTOpE
A3SelD
RD1DSel
RD2DSel
RD1ESel
RD2ESel
DMWDSel
ALUOutE
ALUOutM
DMRDM
DMRDW
WD
PZPC
stall
clr