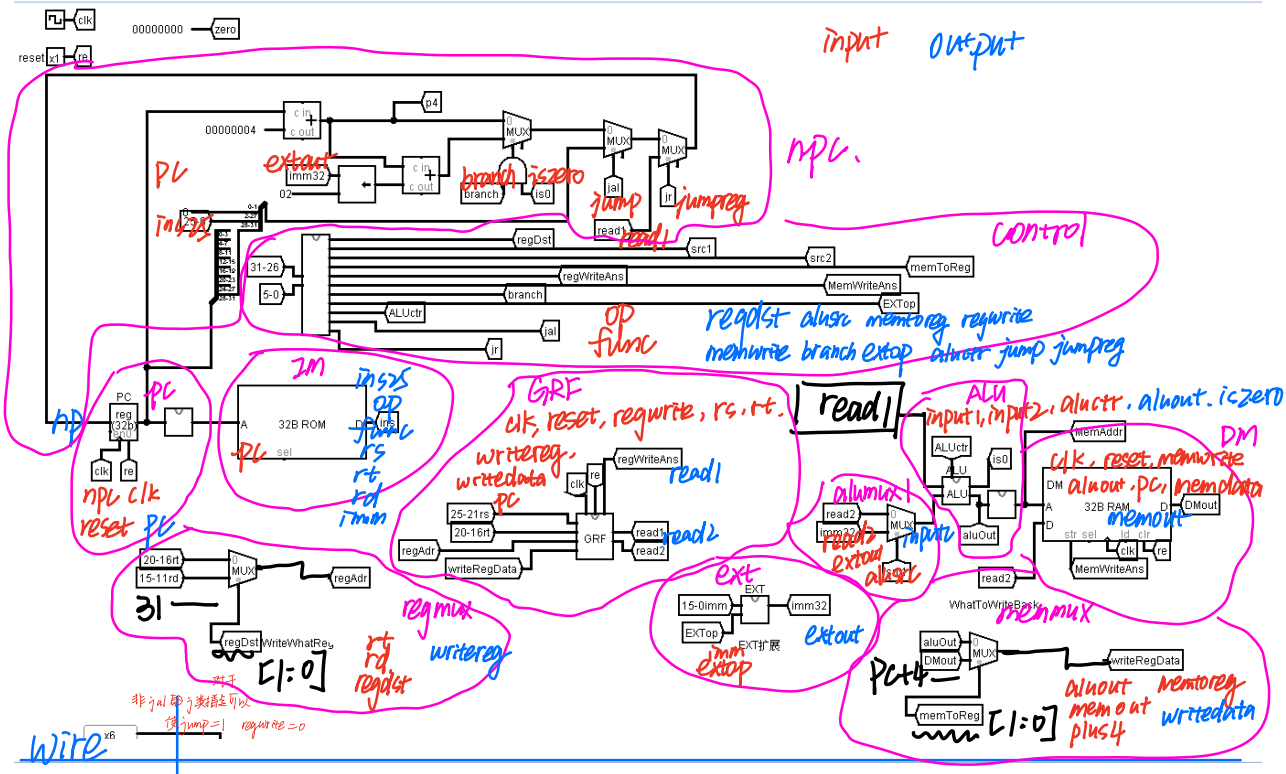


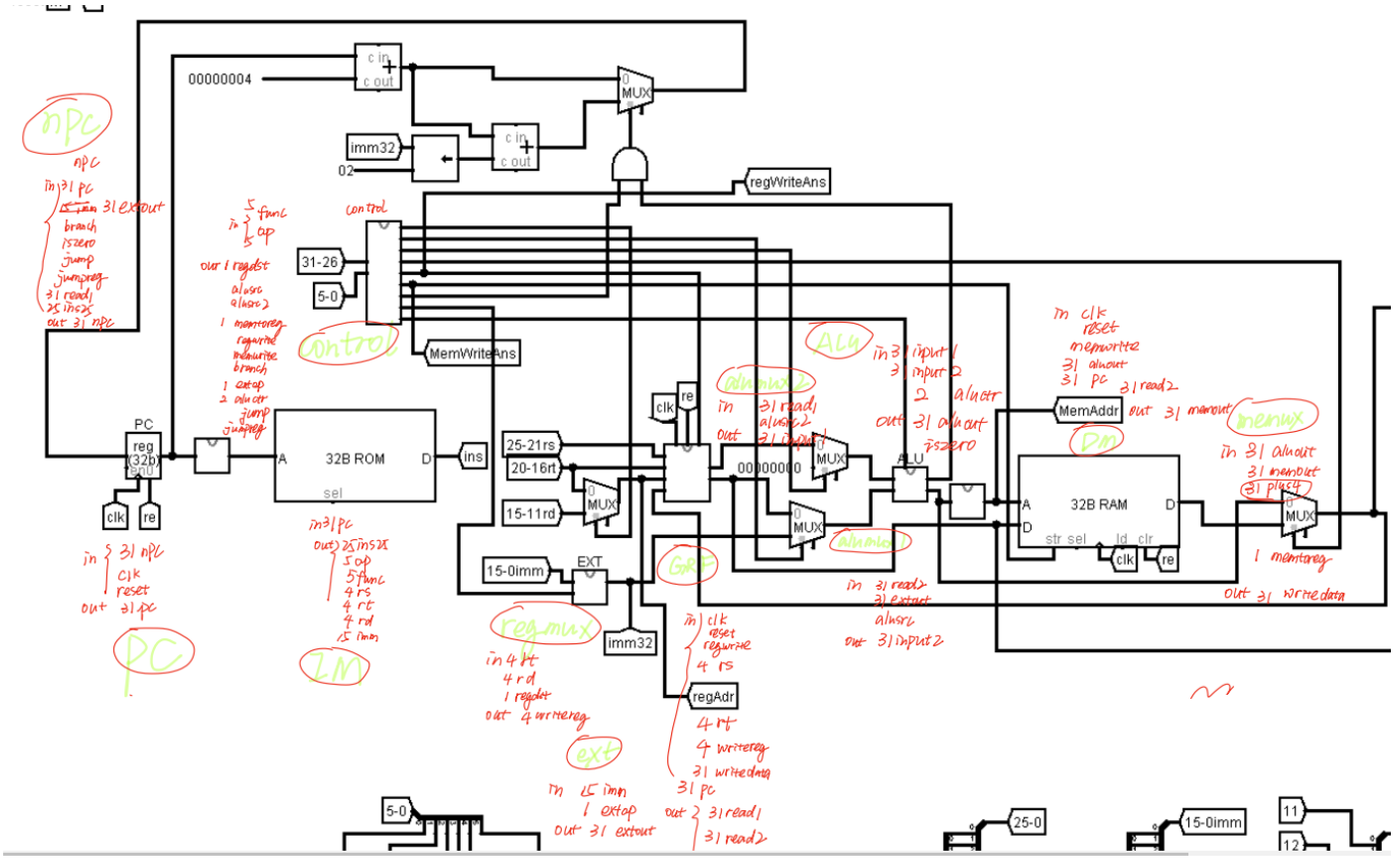
wire

1位.	branch, iszero, jump, jumpreg, alusrc, regwrite, memwrite
[1:0]	regdst, memtoreg, extop
[2:0]	aluctr
[4:0]	rs, rt, rd, writereg,
[5:0]	op, func
[25:0]	ins25
[31:0]	pc, read1, read2, npc, extout, plus4, writedata, input1, input2 aluout memout, memdata

p4设计文档

设计草稿

一、总体



二、部件设计

- 控制信号

名称	add	sub	ori	lw	sw	beq	lui	jal	jr
op（31-26）	100 000	000 000	00 11 01	10 00 11	10 10 11	000 100	00 11 11	000 011	0 0 0 0 0 0

名称	add	sub	ori	lw	sw	beq	lui	jal	jr
func（5-0）	100000	100010							001000
RegDst(有rd的指令需要为1）（mux的选择信号）	01	01	00	00	x	x	00	10(31号寄存器写入）	x
ALUSrc（选rt为0，选立即数为1）（mux的选择信号）	0	0	1	1	1	0	1	x	x
[redacted]									
MemtoReg（选ALU结果为0，选内存DM读出结果为1）（mux的选择信号）	00	00	00	01	x	x	00	10（PC+4写入寄存器）	x
RegWrite（GRF的写入信号）	1	1	1	1	0	0	1	1	0
MemWrite（DM的写入信号）	0	0	0	0	1	0	0	0	0
branch（b类跳转信号）	0	0	0	0	0	1	0	0	0
ExtOp<1:0>(控制拓展类型的信号）(00--0扩展，01--符号扩展，10--加载至高位)	x	x	00	01	01	01	10	x	x
ALUctr<2:0>(ALU运算类型信号）（000--加，001--减，010--or）	000	001	010	000	000	001	000	x	x
j	0	0	0	0	0	0	0	1	0

名称	add	sub	ori	lw	sw	beq	lui	jal	jr
jumpReg	0	0	0	0	0	0	0	0	1

- 模块设计

i. PC

信号名	方向	描述
nextPC<31:0>	I	寄存器输入的值
clk	I	
reset	I	
PCAddress<31:0>	O	PC输出的值

ii. IM

信号名	方向	描述
address<31:0>	I	地址
ins25-0<25:0>	O	
ins31-26op<5:0>	O	
ins5-0func<5:0>	O	
ins25-21rs<4:0>	O	
ins20-16rt<4:0>	O	
ins15-11rd<4:0>	O	
ins15-0imm<15:0>	O	

iii. GRF（寄存器组）

信号名	方向	描述
clk	I	
reset	I	
regwrite	I	
readReg1<4:0>	I	
readReg2<4:0>	I	
writeReg<4:0>	I	
writeData<31:0>	I	

信号名	方向	描述
pc<31:0>	I	为了符合提交的输出
readData1<31:0>	O	
readData2<31:0>	O	

iv. ALU

信号名	方向	描述
input1<31:0>	I	
input2<31:0>	I	
ALUctr<2:0>	I	（000--加，001--减，010--or）
output<31:0>	O	
iszero	O	

v. DM

信号名	方向	描述
clk	I	
reeset	I	
MemWrite	I	
address<31:0>	I	
pc<31:0>	I	为了符合提交的输出
writeData<31:0>	I	
output<31:0>	O	

vi. NPC(组合逻辑)

信号名	方向	描述
PC<31:0>	I	
imm<31:0>	I	还没有左移两位的立即数
branch	I	
iszero	I	
jump	I	
jumpreg	I	

信号名	方向	描述
read1<31:0.>	I	GRF第一个输出值（为了jr指令）
ins25<25:0>	I	
nPC<31:0>	O	

vii. EXT

信号名	方向	描述
in<15:0>	I	
extop<1:0>	I	00零扩展01符号扩展10加载至高位
out<31:0>	O	

viii. MUX（一堆MUX多个module的v文件）

a. regMUX

信号名	方向	描述
rt<4:0>	I	
rd<4:0>	I	
regDst<1:0>	I	00输出rt，01输出rd，10输出31
output<4:0>	O	

b. ALUMUX1

信号名	方向	描述
read2<31:0>	I	
imm<31:0>	I	
ALUsrc	I	0输出read2，1输出imm
output<31:0>	O	

c. ALUMUX2

信号名	方向	描述
read1<31:0>	I	
ALUsrc2	I	0输出read1，1输出32位0
output<31:0>	O	

d. memMUX

信号名	方向	描述
ALUout<31:0>	I	
MEMout<31:0>	I	
addr<31:0>	I	
Memtoreg<1:0>	I(选择信号)	和上面输入的顺序一样选
output<31:0>	O	

测试方案

341c0000
341d0000
34013456
00210820
8c010004
ac010004
3c027878
00411822
3c051234
34040005
00000000
ac85ffff
8c83ffff
10650003
00000000
10000011
00000000
34670404
10e3000e
00000000
3c087777
3508ffff
00080022
34001100
00e65020
34080000
34090001
340a0001
010a4020
1109fffe
0c000c22
00000000
014a5020
1000ffff
014a5020

03e00008
00000000

思考题

1.阅读下面给出的 DM 的输入示例中（示例 DM 容量为 4KB，即 32bit × 1024字），根据你的理解回答，这个 addr 信号又是从哪里来的？地址信号 addr 位数为什么是 [11:2] 而不是 [9:0] ？

是ALU的输出的11到2位构成了address

2.思考上述两种控制器设计的译码方式，给出代码示例，并尝试对比各方式的优劣。

记录指令对应的控制信号取值：

```
case（指令）

case1\:op1=x

&#x20;     op2=x

case2:

endcase
```

记录控制信号对应的取值：

```
 assign op1 = ins1 or ins2 or ins3
```

比较：当指令较多的时候第一种方法可能会更复杂一点，重复太多

3.在相应的部件中，复位信号的设计都是同步复位，这与 P3 中的设计要求不同。请对比同步复位与异步复位这两种方式的 reset 信号与 clk 信号优先级的关系。

同步复位：clk优先

异步复位：reset优先

4.C 语言是一种弱类型程序设计语言。C 语言中不对计算结果溢出进行处理，这意味着 C 语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持 C 语言，MIPS 指令的所有计算指令均可以忽略溢出。请说明为什么在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分

addi在发生溢出的时候会报错，但是addu不会，所以在不考虑溢出的情况下二者等价。add与addu同理。

记录bug

- beq的扩展方法一开始没有考虑

- DM的重置每一位为0一开始不够全面
- nop的控制信号应该在op为000000的else里面，为全0，default是管不到它的