一、模块

1.PC

Ports

Port name	Direction	Type	Description
clk	input		
reset	input		
InPC	input	[31:0]	下一条指令地址
OutPC	output	[31:0]	当前指令地址

2.NPC

Ports

Port name	Direction	Type	Description
Ins	input	[31:0]	当前指令地址
NextIns	output	[31:0]	下一指令地址
Address	input	[25:0]	指令的25-0
Jump	input	[2:0]	是否跳转：0：不跳 1：J 2：Jal 3：Jr
Beq	input		
Ra	input	[31:0]	31号寄存器
Ext	input	[31:0]	

3.IM

Ports

Port name	Direction	Type	Description
RAddr	input	[11:0]	
RData	output	[31:0]	取出的指令

4.Splitter

Ports

Port name	Direction	Type	Description
Ins	input	[31:0]	指令
Op	output	[5:0]	
Rs	output	[4:0]	
Rt	output	[4:0]	
Rd	output	[4:0]	
Shamt	output	[4:0]	
Func	output	[5:0]	
Imme	output	[15:0]	
Address	output	[25:0]	

5.Controller

Ports

Port name	Direction	Type	Description
Ins	input	[31:0]	
Op	input	[5:0]	
Func	input	[5:0]	
RegDst	output	[2:0]	要写入的寄存器
AluSrc	output		进入ALU的数来源：0：rt 1：imme
MemtoReg	output		写入寄存器的数的来源： 0：ALU 1：Mem
RegWrite	output		
MemRead	output		
MemWrite	output		
AluControl	output	[3:0]	

Port name	Direction	Type	Description
AluControl	output	[3:0]	
Sign	output	[1:0]	
branch	output		
jump	output	[2:0]	
MemWriteOp	output	[2:0]	
MemReadOp	output	[2:0]	

6. EXT

Ports

Port name	Direction	Type	Description
Imme	input	[15:0]	
Sign	input	[1:0]	扩展方式
EImme	output	[31:0]	

7. GRF

Ports

Port name	Direction	Type	Description
clk	input		
reset	input		
We	input		写入信号
WPC	input	[31:0]	当前指令
Wd	input	[31:0]	写入的数据
Rs	input	[4:0]	
Rt	input	[4:0]	
Rd	input	[4:0]	写入的地址
RdRs	output	[31:0]	
RdRt	output	[31:0]	

8.ALU

Ports

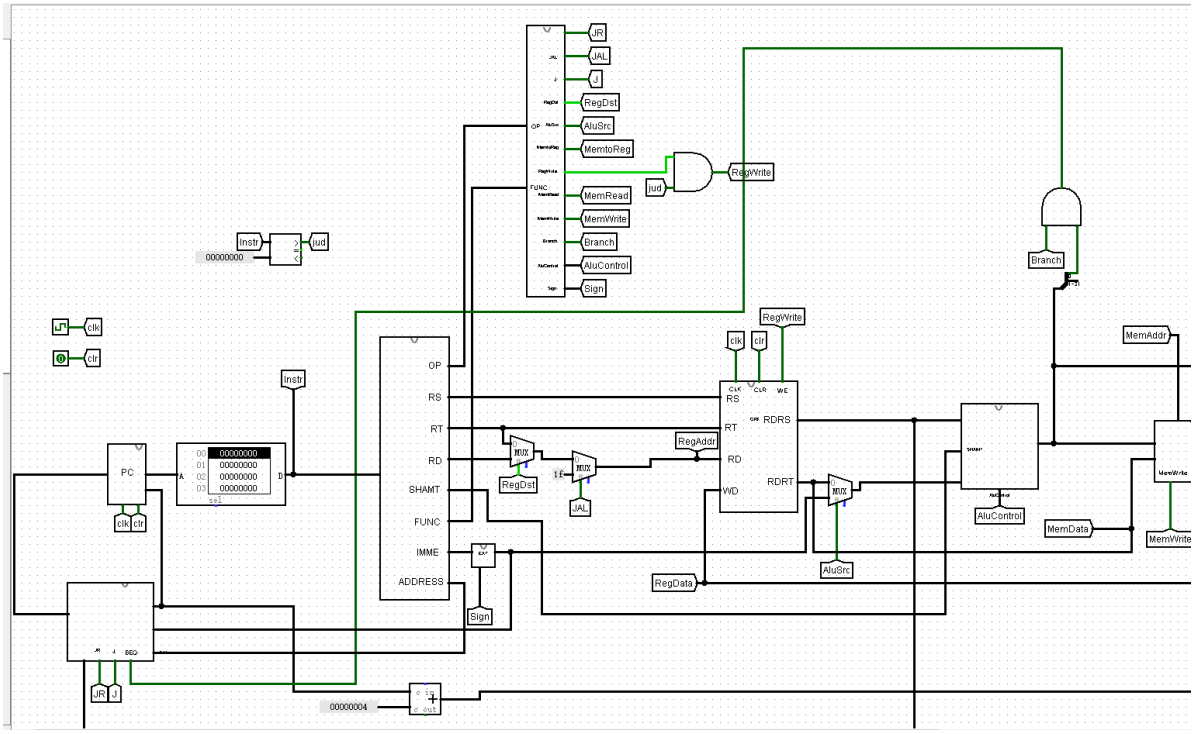
Port name	Direction	Type	Description
A	input	[31:0]	
B	input	[31:0]	
Aluop	input	[3:0]	
Result	output	[31:0]	

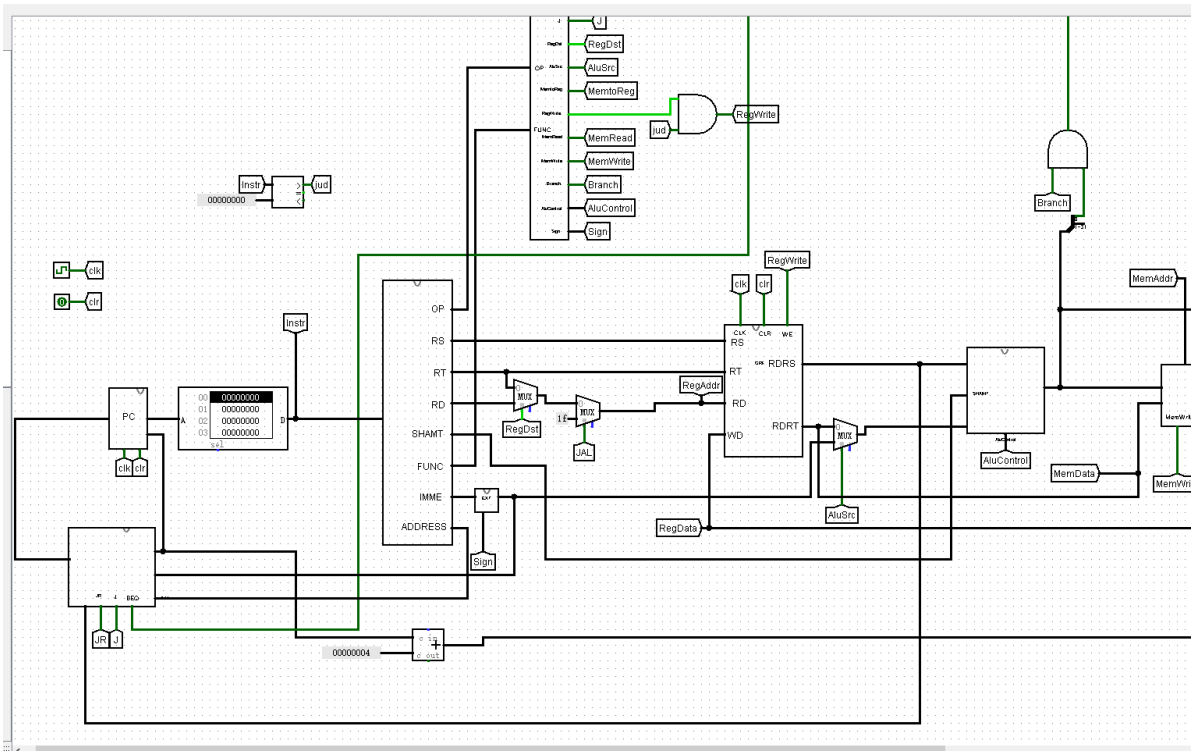
9. DM

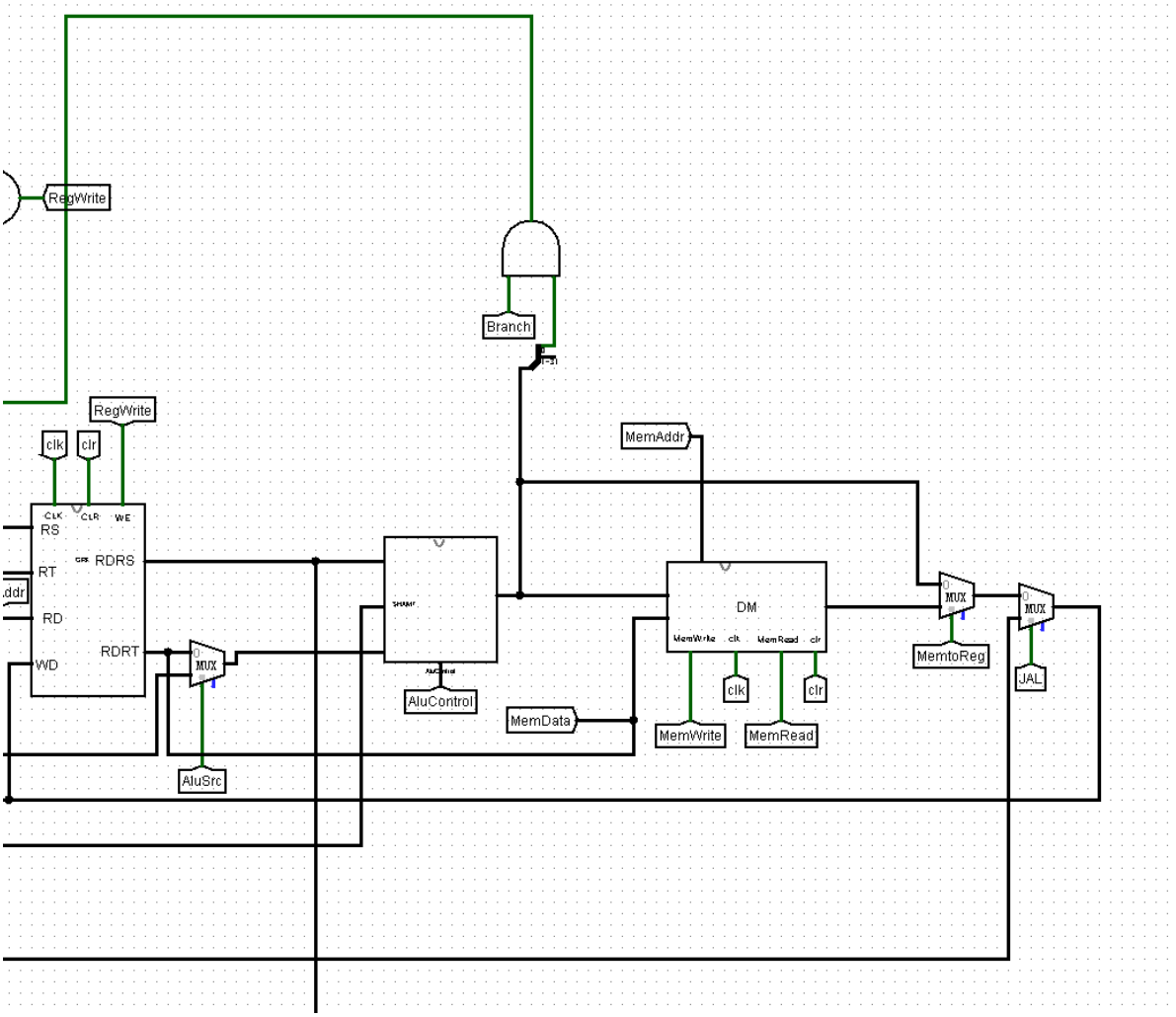
Ports

Port name	Direction	Type	Description
pc	input	[31:0]	
clk	input		
reset	input		
MemWrite	input		
MemRead	input		
MemData	input	[31:0]	
MemAddr	input	[31:0]	
MemWriteOp	input	[2:0]	
MemReadOp	input	[2:0]	
MemOut	output	[31:0]	

二、电路图


JR
JAL
J
RegDst
AluSrc
MemtoReg
RegWrite
MemRead
MemWrite
Branch
AluControl
Sign
Instr
iud
00000000
clk
clr
PC
OP
RS
RT
RD
SHAMT
FUNC
IMME
ADDRESS
MUX
RegAddr
RDRS
RDRT
WD
RegData
AluSrc
MemAddr
MemData
MemWrite
00000004


RegDst
AluSrc
MemtoReg
RegWrite
MemRead
MemWrite
Branch
AluControl
Sign
Instr
iud
00000000
clk
clr
PC
OP
RS
RT
RD
SHAMT
FUNC
IMME
ADDRESS
MUX
RegAddr
RDRS
RDRT
WD
RegData
AluSrc
MemAddr
MemData
00000004



三、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] ？

ALU,字对齐

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。

用add 举例，

第一种：6'b100000:begin

control = 24'b001_0_0_1_0_0_0000_00_0_000_000_000; //add

end

assign {RegDst,AluSrc,MemtoReg,RegWrite,MemRead,MemWrite,AluControl,Sign,branch,jump,MemWriteOp,MemReadOp} = control;

也可以单独列出来，比较清晰

第二种：add

RegDst = add | ...;

AluSrc =

MemtoReg =

RegWrite = add|...

MemRead =

MemWrite =

AluControl =

Sign =

branch =

jump =

MemWriteOp =

MemReadOp =

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。

add除了比addu多一个溢出判断，产生异常意外，其余的计算与addu相同。

addi和addiu同理。

四、测试方案

指令之间没有必然的关联性

逐条测试

ori $0,$0,0x4454

测试是否会改变0号寄存器的值

ori $7,$0,0x1454

测试指令执行是否正确

ori $7,$0,0xf454

测试是否为0扩展

lui $0,0x4785

测试是否会改变0号寄存器的值

lui $7,0x1454

ori $8,$0,0x1454

lui $8,0x1454

测试指令执行是否正确

ori $7,$0,0x1454

ori $8,$0,0x1454

add $0,$7,$8

add $4,$7,$8

sub $0,$7,$8

sub $4,$7,$8

测试加减法

ori $7,$0,0x1454

sw $7,0,($0)

ori $8,$0,4

lui $11,0x478
add $7,$7,$11

sw $7,0,($8)

sw $7,8,($0)

lw $9,0($0)

lw $10,0($8)

非跳转指令测试完成

ori $t0,$0,0x4

ori $t1,$0,0x4

ori $t2,$0,0x7

beq $t0,$t1,jump

add $t0,$t1,$t0

add $t0,$t0,$t0

jump:

sw $t0,0($0)

测试跳转以及跳转位置是否正确

ori $t0,$0,0x4

ori $t1,$0,0x4

ori $t2,$0,0x7

beq $t0,$t2,jump

add $t0,$t1,$t0

add $t0,$t0,$t0

jump:

sw $t0,0($0)

不跳转的情况是否正确

ori $t0,$0,0x4

ori $t1,$0,0x4

jal jump

add $t0,$t0,$t1

add $t1,$t0,$t1

jal end

jump:

sw $t0,0($t2)
ori $t2,$0,4
jr $ra

end:

五、指令

	ADDER	ADDER	PC	IM	GRF	GRF	GRF	GRF	ALU	ALU	DM	DM	SEXT	UEXT	HIGH	SHIFTER	JA		
	A	B			REG1	REG2	WREG	WDATA	A	B	ADDRESS	Write							
add	PC	4	ADDER	PC	[25:21]	[20:16]	[15:11]	ALU	[REG1]	REG[2]	/	/	/	/					
sub	PC	4	ADDER	PC	[25:21]	/	[20:16]	ALU	[REG1]	UEXT	/	/	/	[15:0]					
ori	PC	4	ADDER	PC	[25:21]	[20:16]	[15:11]	ALU	[REG1]	REG[2]	/	/	/	/					
lw	PC	4	ADDER	PC	[25:21]	/	[20:16]	DM	[REG1]	SEXT	ALU	/	[15:0]	/					
sw	PC	4	ADDER	PC	[25:21]	/	[20:16]	/	[REG1]	SEXT	ALU	[20:16]	[15:0]	/					
beq	PC+4	shift	ADDER	PC	[25:21]	[20:16]	/	/	[REG1]	[REG2]	/	/	[15:0]	/	/	SEXT			
lui	PC	4	ADDER	PC	/	/	[20:16]	ALU	0	HIGH	/	/	/	/	[15:0]				
nop																			
J	PC	JA	ADDER	PC													[25:0]		
Jal	PC[31:28]||[25:0]||00		ADDER		/	/	/	PC(now)+4	/	/	/	/	/	/	/	/	/		
Jr	[25:21]	/	ADDER	PC	/	/	/	/	/	/	/	/	/	/	/	/	/		

verilog语法： &按位与 &&逻辑与

|按位或 ||逻辑或

~按位取反

^按位异或

^~同或

所见运算 &a，|a等等， ^a可以用来判断a中的1的个数的奇偶，奇数为1