CPU设计文档

- ctrl模块
  - ctrl.v
    - 端口定义
    - 控制信号
- Datapath模块
  - grf.v
  - alu.v
    - 端口定义
    - ALU控制码
  - ext.v
    - 端口定义
    - 位扩展控制码
  - im.v
    - 端口定义
  - dm.v
    - 端口定义
  - npc.v
    - 端口定义
    - 跳转控制信号
  - mux.v
    - 端口定义
- 思考题
- CPU_Logisim

CPU设计文档

ctrl模块

ctrl.v

端口定义

名称	描述	位宽	方向
op	指令操作码	6	I
funct	指令功能码	6	I
RegDst	GRF写地址选择信号	2	O
ALUSrc	ALU操作数B选择信号	1	O
MemtoReg	寄存器写数据选择信号	2	O
RegWrite	寄存器写使能	1	O
MemWrite	数据存储器写使能	1	O
nPC_sel	跳转控制信号	2	O
ALUOp	ALU控制码	4	O
ExtOp	位扩展控制码	2	O

控制信号

	add	sub	ori	lw	sw	beq	lui	jal	jr
RegDst	01	01	00	00	X	X	00	10	X
ALUSrc	0	0	1	1	1	0	1	X	X
MemtoReg	00	00	00	01	X	X	00	10	X
RegWrite	1	1	1	1	0	0	1	1	0
MemWrite	0	0	0	0	1	0	0	0	0
nPC_sel	00	00	00	00	00	01	00	10	11
ALUctr	ADD	SUB	OR	ADD	ADD	SUB	ADD	X	X
ExtOp	X	X	Zero	Sign	Sign	Sign	High	X	X

Datapath模块

grf.v

名称	描述	位宽	方向
clk	时钟信号	1	I
reset	同步复位信号	1	I
RegWrite	寄存器写使能	1	I
PC	当前指令的地址	32	I
A1	读地址1	5	I
A2	读地址2	5	I
A3	读地址3	5	I
WD	写数据	32	I
RD1	输出A1指定的寄存器数据	32	O
RD2	输出A2指定的寄存器数据	32	O

alu.v

端口定义

名称	描述	位宽	方向
A	操作数A	32	I
B	操作数B	32	I
shamt	移位数	5	I
ALUOp	ALU控制码	4	I
C	运算结果	32	O
isZero	零判断	1	O

ALU控制码

ALUop	Function
0000	ADD
0001	SUB
0010	OR
0011	AND
0100	XOR

ext.v

端口定义

名称	描述	位宽	方向
imm16	16位立即数	16	I
ExtOp	位扩展控制码	2	I
Result	位扩展结果	32	O

位扩展控制码

ExtOp	Function
00	Zero
01	Sign
10	High
11	?

im.v

端口定义

名称	描述	位宽	方向
PC	当前指令地址	32	I
instr	取出的指令	32	O

dm.v

端口定义

名称	描述	位宽	方向
clk	时钟信号	1	I
reset	同步复位信号	1	I
MemWrite	数据存储器写使能	1	I
PC	当前指令地址	32	I
A	读地址	10	I
WD	写数据	32	I
RD	输出A指定地址的数据	32	O

npc.v

端口定义

名称	描述	位宽	方向
PC	当前指令地址	32	I
imm26	26位立即数	26	I
EXT	位扩展器结果	32	I
ra	$ra寄存器的值	32	I
nPC_sel	跳转控制信号	2	I
isZero	零判断信号	1	I
NPC	下一条指令地址	32	O

跳转控制信号

nPC_sel	NPC
00	PC + 4
01	(isZero == 1'b1) ? PC + 4 + (EXT << 2) : PC + 4
10	{PC[31:28], imm26 << 2}
11	GPR[rs]

mux.v

端口定义

- mux_A3

名称	描述	位宽	方向
RegDst	寄存器写地址选择信号	2	I
rt	源操作数寄存器1	5	I
rd	源操作数寄存器2	5	I
A3	寄存器写地址	5	O

- mux_ALUB

名称	描述	位宽	方向
ALUSrc	ALU操作数B选择信号	1	I
RD2	寄存器读数据2	32	I

名称	描述	位宽	方向
imm	经位扩展后的立即数	32	I
B	ALU操作数B	32	O

- mux_grf_WD

名称	描述	位宽	方向
MemtoReg	寄存器写数据选择信号	2	I
C	ALU运算结果	32	I
RD	数据存储器读数据	32	I
PC4	PC + 4	32	I
WD	GRF写数据	32	O

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] ？

文件	模块接口定义

dm.v	dm(clk,reset,MemWrite,addr,din,dout);
	input clk; //clock
	input reset; //reset
	input MemWrite; //memory write enable
	input [11:2] addr; //memory's address for write
	input [31:0] din; //write data
	output [31:0] dout; //read data

addr信号来源于指令。

在DM中，我们需要按字寻址，而MIPS架构中是按字节寻址的，所以应取addr[11:2]而不是[9:0]。

*2.思考上述两种控制器设计的译码方式（记录下**指令对应的控制信号如何取值**或记录下**控制信号每种取值所对应的指令**），给出代码示例，并尝试对比各方式的优劣。*

- 记录下指令对应的控制信号如何取值

```
case (op)
    lw : begin
        assign RegDst = 2'b00;
        assign ALUSrc = 1'b1;
        assign MemtoReg = 2'b01;
        assign RegWrite = 1'b1;
        assign MemWrite = 1'b0;
        //......
    end

    sw : begin
```

```
        assign RegDst = 2'b00;
        assign ALUSrc = 1'b1;
        assign MemtoReg = 2'b00;
        assign RegWrite = 1'b0;
        assign MemWrite = 1'b1;
        //......
    end
endcase
```

这种方法可以清晰地看到每条指令所对应的各个控制信号的取值。但如果有新增的控制信号，需要在每条指令下都添加新增控制信号的取值情况，当指令数量较多时会很不方便。

- 记录下控制信号每种取值所对应的指令

```
    assign RegDst = (op == special && (funct == add || funct == sub)) ? 2'b01 :
                    (op == jal) ? 2'b10 :
                    2'b00;
    assign ALUSrc = (op == ori || op == lw || op == sw || op == lui) ? 1'b1 :
                    1'b0;
    assign MemtoReg = (op == lw) ? 2'b01 :
                      (op == jal) ? 2'b10 :
                      2'b00;
    assign RegWrite = (op == ori || op == lw || op == lui || op == jal) ? 1'b1 :
                      (op == special && (funct == add || funct == sub)) ? 1'b1 :
                      1'b0;
    assign MemWrite = (op == sw) ? 1'b1 :
                      1'b0;
    assign nPC_sel = (op == beq) ? 2'b01 :
                     (op == jal) ? 2'b10 :
                     (op == special && funct == j) ? 2'b11 :
                     2'b00;
    assign ALUOp = (op == beq) || (op == special && funct == sub) ? 4'b0001 :
                   (op == ori) ? 4'b0010 :
                   4'b0000;
    assign ExtOp = (op == lw || op == sw || op == beq) ? 2'b01 :
                   (op == lui) ? 2'b10 :
                   2'b00;
```

这种方法可以清晰地看到每条控制信号在不同取值时对应的指令都有哪些，且新增控制信号较为方便。但在查看某条指令对应的控制信号时并不直观。

*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：**只有在clk上升沿**才会判断reset信号是否有效，其余情况下无论reset是否有效，都不会进行复位；

异步复位：**在任何时候**只要reset有效，无论clk为何值，立即进行复位操作。

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 。

指令集中显示，ADDU(ADDIU)是直接将运算结果存入rd(rt)对应的寄存器中，而ADD(ADDI)是先判断是否溢出，若溢出则抛出异常，否则将结果存入rd(rt)对应的寄存器中。

如果忽略溢出，那么ADDU(ADDIU)和ADD(ADDI)产生的结果均为将运算结果存入rd(rt)对应的寄存器中，所以是等价的。

CPU_Logisim