Logisim单周期CPU设计文档

(一)CPU设计要求

32位单周期CPU

支持指令{add，sub，lw，sw，lui，beq，nop，jr，jal，j}

(二)关键模块设计

ALU

信号名	方向	描述
SrcA[31:0]	I	32位输入信号，第一个操作数A
SrcB[31:0]	I	32位输入信号，第二个操作数B
ALUControl[2:0]	I	3位输入信号，选择ALU的功能 000 : SrcA + SrcB 001 : SrcA - SrcB 010 : A | B 011 : A & B 100 : A>>B 101 : $signed(A)>>>B 110 : A<B 置1
Equal	O	1位输出信号，标志A,B是否相等 A=B置Equal=1
Less	O	1位输出信号，标志A是否小于B A<B置Less=1
ALUResult[31:0]	O	32位输出信号，输出运算结果

GRF

模块定义

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号，将32个寄存器中的值全部清零 1：复位 0：无效
WE	I	写使能信号 1：可向GRF中写入数据 0：不能向GRF中写入数据

信号名	方向	描述
A1[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的值读出到RD1
A2[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的值读出到RD2
A3[4:0]	I	5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器
WD[31:0]	I	32位数据输入信号
RD1[31:0]	O	输出A1指定的寄存器的32位数据
RD2[31:0]	O	输出A2指定寄存器中的32位数据

功能定义

序号	功能名称	描述
1	复位	reset信号有效是，所有寄存器存储的数值清零
2	读数据	读出A1，A2地址对应寄存器中所存储的数据到RD1，RD2
3	写数据	当WE有效且时钟上升沿来临时，将WD写入A3对应的寄存器中

DM

模块定义

信号名	方向	描述
clk	I	时钟信号
Reset	I	复位信号，将RAM中的值全部清零 1：复位 0：无效
WE	I	写使能信号 1：可向GRF中写入数据 0：不能向GRF中写入数据
WD[31:0]	I	32位数据输入信号，要写入的数据
A[31:0]	I	32位输入信号，指定RAM中的的一个地址
DMop[1:0]	I	2位输入信号，用于lb，lh等特殊指令 00： 正常读写，lw，sw 01： 用于lh和sh，根据WE选择进行哪一条指令 10： 用于lb和sb，根据WE选择进行哪一条指令
RD[31:0]	O	32位输出信号，读出A指定的地址中的数据

功能定义

序号	功能名称	描述

序号	功能名称	描述
1	复位	reset信号有效是，RAM所有地址存储的数值清零
2	读数据	读出A指定的地址的所存储的数据到RD
3	写数据	当WE有效且时钟上升沿来临时，将WD写入A对应的地址中

EXT

信号名	方向	描述
Imm[15:0]	I	15位输入立即数
EXTop	I	功能选择信号 0：Imm无符号拓展到32位 1：Imm符号位拓展到32位
EXTImm[31:0]	O	32位输出信号，输出Imm拓展之后的数

NPC

模块定义

信号名	方向	描述
PC[31:0]	I	32位输入信号，当前指令的地址
NPCop[1:0]	I	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31...28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
instr_index[26:0]	I	26位输入信号，用于PC的计算
offset[16:0]	I	16位输入信号，PC的偏移量
Reg[32:0]	I	32位输入信号，用于寄存器地址的跳转
Judge	I	一位输入，作为跳转的判断依据
NPC[31:0]	O	32位输出，输出下一条指令的地址

功能定义

序号	功能名称	描述

序号	功能名称	描述
1	计算PC的下一个值	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31…28}$||instr_index||0_2 10： 结合equal,n_equal,less,big,big or equal,less or equal判断是否需要跳转 PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
2	是否跳转判断	Judge=1，跳转 Judge=0，不跳转

IFU

模块定义

信号名	方向	描述
clk	I	时钟信号
Reset	I	异步复位信号
NPCop[1:0]	I	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31…28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
instr_index[26:0]	I	26位输入信号，用于PC的计算
offset[16:0]	I	16位输入信号，PC的偏移量
Reg[32:0]	I	32位输入信号，用于寄存器地址的跳转
Judge	I	一位输入，作为跳转的判断依据

信号名	方向	描述
Instr[31:0]	O	32为输出信号，输出当前要执行的指令
PC[31:0]	O	32位输出信号，当前PC的地址

CTRL

模块定义

信号名	方向	描述
OP[5:0]	I	6位输入信号，指令操作码

信号名	方向	描述
Func[6:0]	I	6位输入信号，指令的func段
RegDst[1:0]	O	GRFA3输入端控制信号 00 : A3←$Instr_{20...16}$ 01 : A3←$Instr_{15...11}$ 11 : A3←0x1f
RegWrite	O	寄存器写入控制信号 0 : 不能向GRF写入 1 : 可以向GRF写入
EXTop	O	功能选择信号 0：Imm无符号拓展到32位 1：Imm符号位拓展到32位
ALUsrc[1:0]	O	ALUSrcB输入控制信号 00 : SrcB←RD2 01 : SrcB←EXTImm 10 : SrcB←sll指令的s SrcA←RD2
ALUctrl[2:0]	O	3位输出信号，选择ALU的功能 000 : SrcA + SrcB 001 : SrcA - SrcB 010 : A |B 011 : A & B 100 : A>>B 101 : $signed(A)>>>B 110 : A<B 置1 111 : A<<B
Menwrite	O	内存写入控制信号 0 : 不能向DM写入 1 : 可以向DM写入
MemtoReg[1:0]	O	控制向寄存器的写入数据 00 : WD←ALUResult 01 : WD←RD 10 : WD←$NPC_{31...0}$ 11 : WD←[$ALUResult_{15...0}$||0_{16}]

信号名	方向	描述
NPCop[1:0]	O	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31...28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]

信号名	方向	描述
CMPop[2:0]	O	用于指示进行何种跳转判断 000 : 判断equal 001 : 判断n_equal 010 : 判断less 011 : 判断big 100 : 判断less or equal 101 : 判断big or equal
DMop[1:0]	O	用于lb,sb,lh,sh等操作的拓展 00 : 正常读取，以字为单位 01 : 用于lh或sh Menwrite = 1--->sh Menwrite = 0--->lh 11 : 用于lb或sb Menwrite = 1--->sb Menwrite = 0--->lb

CMP

信号名	方向	描述
RD1[31:0]	I	32位输入，作为要比较的值
RD2[31:0]	I	32位输入，作为要比较的值
CMPop[2:0]	I	用于指示进行何种跳转判断 000 : 判断equal 001 : 判断n_equal 010 : 判断less 011 : 判断big 100 : 判断less or equal 101 : 判断big or equal
Judge	O	一位输出，作为跳转的判断依据

相关数据通路信号说明

RegDst

控制对于A3的输入信号，RegDst=00，A3←$Instr_{16...20}$

，RegDst=01，A3←$Instr_{11...15}$

，RegDst=10，A3←0x1f，用于jal指令

ALUsrc

控制对于SrcA的输入信号，ALUsrc=10，SrcA←RD2，用于sll指令

，ALUsrc=other，SrcA←RD1

控制对于SrcB的输入信号，ALUsrc=00，SrcB←RD2

，ALUsrc=01，SrcB←EXTImm

，ALUsrc=10，SrcB←$Instr_{6...10}$(shamt)

MemtoReg

控制对于寄存器的写入信号，MemtoReg=00，WD←ALUResult

，MemtoReg=01，WD←RD

，MemtoReg=10，WD←PC+4

，MemtoReg=11，WD←$ALUResult_{15...0}||0_{16}$，用于lui指令

指令控制信号

Instr	RegDst[1:0]	Regwrite	EXTop	ALUsrc[1]	ALUsrc[0]	ALUctrl[2:0]	Memwrite	MemtoReg[1:0]	NPCop[1:0]	CMPop[2:0]	DMop[1:0]
add	0 1	1	x	0	0	0 0 0	0	xx	0 0	xxx	xx
sub	0 1	1	x	0	0	0 0 1	0	xx	0 0	xxx	xx
ori	0 0	1	0	0	1	0 1 0	0	xx	0 0	xxx	xx
lw	0 0	1	1	0	1	0 0 0	0	0 1	0 0	xxx	0
sw	xx	0	1	0	1	0 0 0	1	xx	0 0	xxx	0
beq	xx	0	x	0	0	x x x	x	xx	1 0	0 0 0	xx
lui	0 0	1	0	0	1	0 0 0	x	1 1	0 0	xxx	xx
jal	1 0	1	x	x	x	xxx	x	1 0	0 1	xxx	xx
jr	0 0	0	x	x	x	xxx	0	xx	11	xxx	xx
j	xx	0	x	x	x	xxx	0	xx	0 1	xxx	xx

(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]？

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

这个addr信号是ALU的结果，作为DM的地址，由于DM是以字编址，每四个字节编一个地址，而ALUresult计算出来的地址是以字节编址，所以换算成字地址需要每四位取一位地址，即从ALUresult的第三位开始取地址，因为DM的容量是4KB，所以按照字节编址是12位地址，取ALUresult[11:0]，以字做单位就是10位地址，便是取10位，即[11:2]，同时以[11:2]作为地址可以提醒使用者，给DM的addr引脚接入的是ALUresult的[11:2]位。

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。

指令对应的控制信号如何取值

```
`define R 6'b000000
·define lw 6'b100011
case(Instr[31:26])
    `R:
        begin
            case(Instr[5:0])
                6'b100000://add
                    RegDst=2'b01;
                    Regwrite=1'b1;
                    ALUsrc=2'b00;
                    ALUctrl=3‘b000;
                    Memwrite=0;
                    MemtoReg=2'b00;
                6'b100010://sub
                    RegDst=2'b01;
                    Regwrite=1'b1;
                    ALUsrc=2'b00;
                    ALUctrl=3‘b001;
                    Memwrite=0;
                    MemtoReg=2'b00;
                ……
            endcase
        end
      `lw:
          ……
endcase
```

这一种译码方式，对于信号的控制不容易遗漏，对于每一个信号都需要给一个值，清晰易读比较直观，但是添加指令比较复杂，需要给出完整的控制信号，对于不需要的信号也需要给定默认值，这样也会导致控制部分的代码比较长。

控制信号每种取值所对应的指令

```
wire add=(Instr[31:26]==6'b000000 && Instr[5:0] == 6'b100000);
wire sub=(Instr[31:26]==6'b000000 && Instr[5:0] == 6'b100010);
……
    assign RegDst[1]=1'b0|jal;
    assign RegDst[0]=1'b0|add|sub;
    assign Regwrite=1'b0|add|sub|ori|lw|lui|jal;
    assign EXTop=1'b0|lw|sw;
    assign ALUsrc[1]=1'b0;
    assign ALUsrc[0]=1'b0|ori|lw|sw|lui;
    assign ALUctrl[2]=1'b0;
    assign ALUctrl[1]=1'b0|ori;
    assign ALUctrl[0]=1'b0|sub;
    assign Memwrite=1'b0|sw;
    assign MemtoReg[1]=1'b0|lui|jal;
    assign MemtoReg[0]=1'b0|lw|lui;
    assign NPCop[1]=1'b0|beq|jr;
```

```
    assign NPCop[0]=1'b0|jal|jr|j;
    assign Branchop[2]=1'b0;
    assign Branchop[1]=1'b0;
    assign Branchop[0]=1'b0;
    assign DMop[1]=1'b0;
    assign DMop[0]=1'b0;
```

这一种译码方式，是对控制信号用了或指令的方式，如果满足这条指令，就会使的控制信号有效，这种方式的优点在于可以很容易的添加指令，对于指令只需要在相应控制信号之后或上一个即可，但是缺点是不够直观，可能会造成漏加信号的错误。

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。

在同步复位中，clk的优先级是高于reset的，只有在时钟上升沿到来时reset信号有效才进行复位，单有reset信号而时钟上升沿信号没有到来不能进行复位。以verilog代码来说，reset信号是在clk的触发中的，只有外层有效内层才能有效。

```
always@(posedge clk)begin
    if(reset==1)begin
    ……
    end
end
```

在异步复位中，clk和reset的优先级是相同的，不论是clk和reset都能触发控制，即在任何时候只要reset信号有效，都能触发复位，以verilog代码来说，敏感条件是posedge clk或者posedge reset，只要二者满足其中一个情况，就可以进行接下来的操作。

```
always@(posedge clk or posedge reset)begin
    if(reset==1)begin
    ……
    end
end
```

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与addiu的区别在于，当出现溢出时，addiu将忽略溢出，将溢出的最高位舍弃；addi会报告SignalException(IntegerOverflow)。

如果忽略溢出，addi不会报错，二者等价。add和addu同理。