CPU设计文档

一、数据通路设计

1.IFU

a.模块接口

表 1

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| nPC\_sel | I | beq指令判断信号  0:非beq指令  1:是beq指令 |
| zero | I | R[rs] 和R[rt]大小比较信号  0: R[rs] != R[rt]  1: R[rs] = R[rt] |
| Imm16[15:0] | I | 16位立即数 |
| Reset | I | 同步复位信号  1：复位  0：无效 |
| Clk | I | 时钟信号 |
| j\_instr | I | jal指令判断信号  0:非jal指令  1:是jal指令 |
| jr | I | jr指令判断信号  0:非jr指令  1:是jr指令 |
| RD1[31:0] | I | R[rs] |
| Imm26[25:0] | I | 26位立即数 |
| Instr[31:0] | O | 32位指令 |
| PC[31:0] | O | 当前PC值 |

b.功能定义

表 2

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算NPC | 若为jal指令：PC←PC[31:28]||imm26||02  若为jr指令：PC←RD1  若nPC\_sel和zero均为1：  PC←PC+4+signed\_ext(imm||02)  其他：  PC←PC+4 |
| 2 | 取出指令Instr | 根据PC寄存器的地址从ROM取出对应指令 |
| 3 | 复位 | 时钟上升沿若复位信号若有效则将PC置为0x00003000 |

2.GRF

a.模块接口

表 3

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A1[4:0] | I | 读入的第一个寄存器的地址 |
| A2[4:0] | I | 读入的第二个寄存器的地址 |
| A3[4:0] | I | 写入的寄存器序号 |
| WD[31:0] | I | 写入寄存器堆的32位信号 |
| 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 |
| Clk | I | 时钟信号 |
| Reset | I | 同步复位信号  1：复位  0：无效 |
| PC[31:0] | I | 当前执行的指令对应的PC值 |
| RD1[31:0] | O | 输出的A1对应的寄存器的32位数据 |
| RD2[31:0] | O | 输出的A2对应的寄存器的32位数据 |

b.功能定义

表 4

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 读出A1、A2地址对应寄存器中存储的数据到RD1,RD2 |
| 2 | 写数据 | 时钟上升沿到来时若WE有效，将WD写入A3对应的寄存器中 |
| 3 | 复位 | 时钟上升沿到来时若Reset信号有效，所有寄存器的数值被清零 |

3.ALU

a.模块接口

表 5

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 参与计算的第一个32位数据 |
| B[31:0] | I | 参与计算的第二个32位数据 |
| ALUOp[1:0] | I | 计算类型  00：加法  01：减法  10：或运算 |
| C[31:0] | O | 计算得到的32位数据 |

b.功能定义

表 6

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法运算 | C=A+B |
| 2 | 减法运算 | C=A-B |
| 3 | 或运算 | C=A | B |
| 4 | 比较大小 | 当计算类型为减法时，  C>0: A>B  C=0: A=B  C<0: A<B |

4.DM

a.模块接口

表 7

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[9:0] | I | 读或写DM的地址 |
| DataIn[31:0] | I | 写入的32位数据 |
| DMwrite | I | 写使能信号：  0：不进行写入操作  1：进行写入操作 |
| PC[31:0] | I | 当前执行的指令对应的PC值 |
| Clk | I | 时钟信号 |
| Reset | I | 同步复位信号  1：复位  0：无效 |
| DataOut[31:0] | O | 读出对应地址的 32位数据 |

b.功能定义

表 8

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读存储器 | 将Addr对应的RAM的地址中的数据取出并输出 |
| 2 | 写存储器 | 时钟上升沿到来时，若DMwrite有效，将32位数据写入Addr对应的RAM的地址区域 |
| 3 | 复位 | 时钟上升沿到来时若复位信号有效，将RAM的数据清零 |

5.EXT

a.模块接口

表 9

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm16[15:0] | I | 被扩展的16位立即数 |
| EXTOp[1:0] | I | 扩展类型选择信号  00：零扩展  01：符号扩展  10：将立即数加载至高位，低位补零 |
| imm32[31:0] | O | 扩展后的32位立即数 |

b.功能定义

表 10

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 零扩展 | imm32=zero\_ext(imm16) |
| 2 | 符号扩展 | imm32=signed\_ext(imm16) |
| 3 | 加载至高位，低位补零 | imm32=imm16||016 |

6.MUX

a.模块接口

表 11

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| rt[4:0] | I | rt域 |
| rd[4:0] | I | rd域 |
| RegDst[1:0] | I | 寄存器堆写入地址选择信号：  00：写入地址为rt  01：写入地址为rd  10：写入地址为31 |
| R2[31:0] | I | R[rt] |
| imm32[31:0] | I | 经扩展后的32位立即数 |
| ALUSrc | I | ALU第二个运算值选择信号  0：R[rt]参与运算  1：imm32参与运算 |
| ALUans[31:0] | I | ALU运算得到的结果 |
| Memdout[31:0] | I | 数据存储器读出的数据 |
| PC[31:0] | I | 当前指令的PC值 |
| MemtoReg[1:0] | I | 寄存器堆写入数据的选择信号  00：把ALU运算结果写入  01：把数据存储器读出的数据写入  10：把PC+4写入 |
| RegAddr[4:0] | O | 寄存器堆写入地址 |
| ALUsec[31:0] | O | ALU第二个运算值 |
| RegData[31:0] | O | 寄存器堆写入数据 |

b.功能定义

表 12

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择寄存器堆写入地址 | 根据控制信号RegDst选择寄存器堆写入地址 |
| 2 | 选择ALU第二个运算值 | 根据控制信号ALUSrc选择ALU第二个运算值 |
| 3 | 选择寄存器堆写入数据 | 根据控制信号MemtoReg选择寄存器堆写入数据 |

二、控制器设计

1.真值表

表 13

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | 10 0001 | 10 0011 |  | | | | | | 001000 |
| op | 000000 | 00 0000 | 00 1101 | 10 0011 | 10 1011 | 000100 | 00 1111 | 000011 | 000000 |
|  | addu | subu | ori | lw | sw | beq | lui | jal | jr |
| RegDst[1:0] | 01 | 01 | 00 | 00 | X | X | 00 | 10 | X |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 | X | X |
| MemtoReg  [1:0] | 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 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| ExtOp[1:0] | X | X | 00 | 01 | 01 | X | 10 | X | X |
| ALUctr[1:0] | 00 | 01 | 10 | 00 | 00 | 01 | 00 | X | X |
| j\_instr | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| jr | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

2.控制信号说明

RegDst:寄存器堆写入地址的选择信号（rd or rs or 31）

00：选择rt作为写入地址

01：选择rd作为写入地址

10：选择31作为写入地址

ALUSrc:参与ALU运算的第二个数据的选择信号

0：选择rt对应寄存器的值

1：选择扩展后的立即数

MemtoReg:写入寄存器堆的数据的选择信号

00：选择ALU的输出

01：选择DM的输出

10：选择PC+4

RegWrite:寄存器堆的写使能信号

0：不向寄存器堆写入数据

1：向寄存器堆写入数据

MemWrite:DM的写使能信号

0：不向DM写入数据

1：向DM写入数据

nPC\_sel: beq指令的判断信号

0：不是beq指令

1：是beq指令

ExtOp:16位立即数位扩展选择信号

00：零扩展

01：符号扩展

10:将16位立即数加载至高位，低位补零

ALUctr:ALU运算类型选择信号

00：加法

01：减法

10：或运算

j\_instr: jal指令的判断信号

0：不是jal指令

1：是jal指令

jr: jr指令的判断信号

0：不是jr指令

1：是jr指令

三、测试程序

1.测试代码

loop1:

ori $t0,$0,0x5678

lui $t1,0x1234

ori $t2,$t1,0x4321

ori $t3,$0,2

beq $t1,$t3,loop1

nop

addu $t4,$t3,$t1

sw $t0,0($0)

jal loop2

nop

sw $t1,4($0)

lw $t2,2($t3)

loop3:

beq $t1,$t2,loop3

loop2:

subu $t5,$t0,$t3

addu $t5,$t5,$t3

jr $ra

2.测试期望

@00003000: $ 8 <= 00005678

@00003004: $ 9 <= 12340000

@00003008: $10 <= 12344321

@0000300c: $11 <= 00000002

@00003018: $12 <= 12340002

@0000301c: \*00000000 <= 00005678

@00003020: $31 <= 00003024

@00003034: $13 <= 00005676

@00003038: $13 <= 00005678

@00003028: \*00000004 <= 12340000

@0000302c: $10 <= 12340000

思考题

一、数据通路设计

1.根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？

答：因为在定义DM时为如下代码：reg [31:0] dm\_reg [1023:0]，即DM以字为单位寻址，而mips以字节为单位寻址,1字等于4字节，故取addr的[11:2]；addr信号来自ALU模块的输出，为ALU模块根据基地址和偏移量计算出的结果。

2.在相应的部件中，reset的优先级比其他控制信号（不包括clk信号）都要高，且相应的设计都是同步复位。清零信号reset是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

答：reset将PC复位至0x00003000，将GRF和DM清零；将PC复位以使得测试程序回到最初，重新执行第一条指令；将GRF和DM清零是为了清除之前运行指令写入GRF和DM的数据，使得新的测试不受影响。

二、控制器设计

1.列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。

答：

1.if-else:

if(op==6'b100011)begin

RegDst=2'b00;

ALUSrc=1;

MemtoReg=2'b01;

RegWrite=1;

MemWrite=0;

nPC\_sel=0;

ExtOp=2'b01;

ALUctr=2'b00;

j\_instr=0;

jr=0;

end

2. assign语句：

assign ALUSrc=(op==6'b001101)|(op==6'b100011)|(op==6'b101011)

|(op==6'b001111);

3. 宏定义:

`define lw 6'b100011

2.根据你所列举的编码方式，说明他们的优缺点。

答：if-else语句逻辑表达很清晰，但是代码量比较大且可读性不强；

assign语句代码量较少，采用或逻辑；

宏定义可以使代码的可读性大大增强，有助于debug。

三、在线测试相关信息

1.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 。

答：addi和addiu均是将rs对应的寄存器的值和经符号扩展后的立即数相加，把结果写入rt对应的寄存器，二者唯一区别是addi会检测溢出而addiu不会检测溢出，所以在忽略溢出的前提下二者等价。

2.根据自己的设计说明单周期处理器的优缺点。

答：优点：结构易于解释且控制单元相对简单。

缺点：时钟周期由最慢的指令决定，降低了指令执行的效率，且吞吐量低；另外它采用独立的指令存储器和数据存储器，在实际系统中不太现实。

3.简要说明jal、jr和堆栈的关系。

答：jal和jr一般用于函数调用和返回，jal调用子函数时将PC+4写入$ra，进入子函数后通常将$ra的值写入栈并移动栈指针$sp，在函数调用结束后从栈中取出$ra的值，通过jr指令返回调用函数并执行jal的下一条指令。