Verilog实现的单周期CPU设计文档

一、设计综述

1. 体系结构综述

   处理器为32位简单MIPS单周期CPU，支持指令集为：

   `add, sub, ori, lw, sw, beq, lui, jal, jr, nop`

   指令操作码与MIPS官方设计一致

指令	OPcode	Funct
nop	000000	000000
add	000000	100000
sub	000000	100010
ori	001101	NA
lw	100011	NA
sw	101011	NA
beq	000100	NA
lui	001111	NA
jal	000011	NA
jr	000000	001000

2. 设计思路

   - 模块化层次化设计
   - 顶层驱动信号仅包括异步复位信号

二、关键模块设计

设计思路应当与Logisim文档一致

IFU

- 基本信息

序号	信号名	描述
1	clk	时钟信号
2	rst	异步复位至`0x00000000`
3	beq	检测是否是beq
4	eq	是否相等
5	imm32[31:0]	32位拓展后立即数
6	instr[31:0]	32位指令

序号	功能

序号	功能
1	异步复位到0x00000000
2	正常递增PC<=PC+4
3	在beq条件成立的情况下跳转

注：PC的输出应当无符号加0x3000以指向地址中0x00003000的起始位置。

NPC

NPC是PC次态地址的组合选择逻辑电路。

序号	信号名	描述
1	NPC_signal[1:0]	NPC选择信号
2	exImm	j指令的立即数地址
3	offSet	beq的偏移量
4	jrAddr	jr的地址
5	curPC	当前PC值
6	nextPC(O)	次态PC值

Splitter

序号	信号名	描述
1	instr[31:0]	输入指令
2	opCpde[5:0]	Op字段
3	rs[4:0]	rs寄存器地址
4	rt[4:0]	rt寄存器地址
5	rd[4:0]	rd寄存器地址
6	shamt[4:0]	偏移量
7	funct[5:0]	funct字段
8	offset[15:0]	beq偏移量
9	imm[25:0]	j立即数

ALU 算数逻辑单元

- 基本信息

序号	信号名	描述
1	In1	输入端1
2	In2	输入端2
3	Op[2:0]	操作码
4	eq(O)	比较是否相等
5	zero(O)	输出是否是零

序号	信号名	描述
6	out(O)	输出运算结果

序号	功能
1	根据ALUop进行运算：
000	addu
001	subu
010	and
011	or

GRF 寄存器堆

- 基本信息

序号	信号名	描述
1	clk	时钟信号
2	rst	异步复位信号
3	WE	写使能信号
4	WD	32位写入数据
5	W	5位写寄存器地址
6	R1	5位读寄存器地址
7	R2	5位读寄存器地址
8	R1(O)	R1读寄存器输出
9	R2(O)	R2读寄存器输出

序号	功能
1	从读寄存器读取数据
2	向写寄存器写入数据
3	零寄存器读取写入都不产生任何影响

DM 数据存储器

- 基本功能

序号	信号名	描述
1	clk	时钟信号
2	rst	异步复位信号
3	WE	写使能信号
4	ld	读使能信号
5	WD[31:0]	32位写入数据
6	addr[4:0]	访问地址

控制器设计

Controller 控制器设计

`MainDecoder` + `ALUDecoder` + `jumpDecoder`

`add, sub, ori, lw, sw, beq, lui, jal, jr, nop`:

Index	Instr	Opcode	func	RegWE	RegData	RegDes	ALUsrc	ALUop	DMRE	DMWE	nextPCop
1	add	000000	100000	1	1	1	1	add	No	No	00
2	lw	100011(i)	NA	1	0	0	0	add	Yes	No	00
3	sw	101011(i)	NA	0	x	x	0	add	No	Yes	00
4	sub	000000	100010	1	1	1	1	sub	No	No	00
5	ori	001101(i)	NA	1	1	0	0	or	No	No	00
6	beq	000100(i)	NA	0	x	x	1	sub	No	No	11
7	lui	001111(i)	NA	1	1	0	0	sll	No	No	00
8	jal	000011	NA	1	11	11	x	x	No	No	01
9	jr	000000	001000	0	x	x	x	x	No	No	10

思考题回答

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] ？

地址信号来自于指令与寄存器基址的计算。之所以不是`[9:0]`是因为我们在这里采用的是按字节分配地址，而取字指令应当能够取得整个字，因此必须左移两位。

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。

两种指令设计示例：

- 指令对应信号

op	ALUsrc	ALUop
add	0	000

- 信号对应指令

port	add	sub
ALUsrc	0	0

前者可以清晰直观分析指令对应端口的操作信号，后者可以清晰知晓端口什么时候才应该取有效电平。

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。

同步复位中`clk`信号优先级更高，异步复位则相反。

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。

在忽略溢出的前提下，`addi` 与 `addiu` 等价，`add` 与 `addu` 等价，是因为这些指令的底层加法运算机制完全相同：对于相同的输入操作数，它们都会执行相同的32位二进制加法运算，产生完全一致的32位结果。唯一的区别在于 `addi` 和 `add` 会在有符号溢出时触发异常，而 `addiu` 和 `addu` 则不会进行溢出检查。因此，当程序员确保运算不会溢出或主动忽略溢出时，这些指令在功能上变得完全等效。