使用verilog的单周期MIPS CPU设计文档

0. 综述

本CPU为Logisim实现的单周期CPU，支持的指令集包含`{addu,subu,addi,ori,lw,sw,lb,sb,lui,nop,sll,slti,beq,j,jal,jr,jalr}`。为了实现这些功能，CPU主要包含了IFU、GRF、ALU、DM、Ext、Control、BrCmp、datapath、MUX_4、NPC、settings模块。

一、模块规格

1. IFU模块（取指单元）

端口定义：

端口名	方向	位宽	功能描述
Clk	Input	1	时钟信号
Reset	Input	1	异步复位信号
NPC	Input	[31:0]	下一次PC值
Instruction	Output	[31:0]	当前指令
PCOut	Output	[31:0]	当前PC值

具体功能：

功能	描述
输出下一条指令	在clk上升沿时，① 当PCSrc为0时，$PC \leftarrow PC + 4$; ② 当PCSrc为1时，$PC \leftarrow PC + 4 + sign\_ext(offset||0^2)$
输出当前PC值	
复位	reset信号变为1时，PC清零

2. NPC

端口名	方向	位宽	功能描述
ifBr	I	1	是否满足B类跳转条件
isBr	I	1	是否是Branch指令
isJump	I	1	是否是J类型指令
isJr	I	1	是否是Jr类型指令
BrImm	I	[31:0]	Br类型跳转立即数
JImm	I	[31:0]	J类型跳转立即数
JrImm	I	[31:0]	Jr类型跳转立即数
PC	I	[31:0]	PC值
NPC	O	[31:0]	Next PC值

3. BrCmp

端口名	方向	位宽	功能描述
Instr	I	[31:0]	指令
RData1	I	[31:0]	读入数据1
RData2	I	[31:0]	读入数据2
ifBr	O	1	是否满足B类跳转条件

4. GRF模块（寄存器堆）

端口定义：

端口名	方向	位宽	功能描述
Clk	Input	1	时钟信号
Reset	Input	1	异步复位信号
WE	Input	1	写使能端
ReadAddr1	Input	[4:0]	读寄存器编号1
ReadAddr2	Input	[4:0]	读寄存器编号2
WriteAddr	Input	[4:0]	写寄存器编号
WData	Input	[31:0]	写入数据
RData1	Output	[31:0]	读寄存器值1
RData2	Output	[31:0]	读寄存器值2

具体功能：

功能	描述
复位	当reset为1时，所有寄存器的值清零
读取数据	RData1的值是寄存器编号为Read1的寄存器的值；RData2的值是寄存器编号为Read2的寄存器的值
写入数据	当写使能WE为1时，向编号为Write的寄存器写入WriteData

5. ALU模块（算术逻辑单元）

端口定义：

端口名	方向	位宽	功能描述
A	Input	[31:0]	输入数据1
B	Input	[31:0]	输入数据2
ALUOp	Input	[4:0]	选择ALU功能
Shamt	Input	[4:0]	左移位数
Zero	Output	1	运算结果是否为0
Result	Output	[31:0]	运算结果

具体功能：

功能	描述
与	`Result=A&B`
或	`Result=A|B`
异或	`Result=A^B`
减（比较）	`Result=A-B`；若 `A==B,Zero=0`
加	`Result=A+B`
逻辑左移	`Result=B<<Shamt`

6. DataMemory模块

端口定义：

端口名	方向	位宽	功能描述
Clk	Input	1	时钟信号
Reset	Input	1	异步复位信号
WE	Input	1	写使能信号
Addr	Input	[4:0]	写入地址
WD	Input	[31:0]	写入数据
RD	Output	[31:0]	读取数据

具体功能：

功能	描述
复位	Reset 为 1 时，数据清 0
写入数据	当时钟上升沿时，如果写使能信号WE有效，就将WD写入到地址为Addr处
读取数据	RD为地址Addr处的数据的值

7. Ext（数据扩展器）

端口定义：

端口名	方向	位宽	功能描述
Imm16	Input	[15:0]	输入的16位立即数
EXtCtrl	Input	[1:0]	控制信号
Imm32	Output	[31:0]	扩展结果

具体功能：

功能	描述
0扩展	`Imm32={{16{0}},Imm16}`
符号扩展	`Imm32={{16{Imm16[15]}},Imm16}`
把数加载到高位	`Imm32={Imm16,{16{0}}}`
1扩展	`Imm32={{16{1}},Imm16}

6. datapath

将前述5个模块实例化并连接

端口名	方向	位宽	功能描述
clk	In	1	时钟信号
reset	In	1	异步复位信号
RegDst	In	[1:0]	MUX选择写回寄存器
ALUSrc	In	1	MUX选择ALU端口B的输入
MemtoReg	In	[1:0]	MUX选择写回数据
RegWrite	In	1	GRF写使能
MemWrite	In	1	DM写使能
Branch	In	1	是否是 beq 指令
ExtCtrl	In	[1:0]	扩展器控制
AluOp	In	[4:0]	ALU功能
isJump	In	1	J类型指令
isJR	In	1	JR指令
instr	out	[31:0]	指令数据

二、控制信号

1. 指令编码

	addu	subu	ori	lw	sw	beq	lui	sll
op	000000	000000	001101	100011	101011	000100	001111	000000
func	100001	100011	n/a	n/a	n/a	n/a	n/a	000000
	j	jal	jr					
op	000010	000011	000000					
func	n/a	n/a	001000					

2. 控制信号真值表

	RegDst	RegWrite	EXTCtrl	ALUSrc	ALUOp	WE	MemToReg	Branch	JUMP	JR
作用	MUX选择写回寄存器	GRF写使能	扩展器控制	MUX选择ALU端口B的输入	ALU功能	DM写使能	MUX选择写回数据	是否是beq指令	是否J类跳转	是否JR
addu	1	1	x	0	add	0	0	0	0	0
subu	1	1	x	0	sub	0	0	0	0	0
ori	0	1	00	1	or	0	0	0	0	0
lw	0	1	01	1	add	0	1	0	0	0
sw	x	0	01	1	add	1	x	0	0	0
beq	x	0	x	0	sub	0	x	1	0	0
lui	0	1	10	1	add	0	0	0	0	0
nop	x	0	x	x	x	0	x	0	0	0
sll	1	1	x	0	sll	0	0	0	0	0
j	x	0	sign	1	add	0	x	0	1	0
jal	恒31	1	x	x	x	0	10	0	1	0
jr	x	0	x	0	x	0	0	0	0	1

三、测试CPU

1.

```
ori     $t0, $zero, 1   # t0=1
ori     $t1, $zero, 2   # t1=2
ori     $t3, $zero, 4   # t3=4
ori     $t2, $t0, 1     # t2=1
sw      $t2, 8($zero)
sw      $t1, 12($zero)
lw      $t2, 8($t3)     # t2=2
eq:
lw      $t1, 8($zero)   # t1=1
addu    $t4, $t1, $t0   # t4=2
addu    $1, $1, $t1     # $1=1
sw      $1, 4($zero)
lw      $2, 4($zero)    # $2=1
beq     $t4, $t0, eq    # not equal
subu    $t4, $t4, 1     # t4=1
beq     $t4, $t0, eq    # equal
```

复合指令，主要用于观测指令能否正常运行。运行过程中寄存器堆中的寄存器是否正确存入数据，DM是否正确储存值，PC是否如预期跳转(-6和-10)

2.

```
ori     $t1, $0, 32
ori     $t2, $0, 0
ori     $t0, $0, 0
nop
nop
for_begin:
beq     $t0, $t1, for_end
sw      $t0, 0($t2)
ori     $s0, $0, 4
addu    $t2, $t2, $s0
ori     $s0, $0, 1
addu    $t0, $s0, $t0
beq     $0, $0, for_begin
for_end:
nop
nop
```

期望：DM中地址0~31分别填入0~31

3.

```
# test ori
ori $a0, $0, 123
ori $a1, $a0, 456

# test lui
lui $a2, 123
lui $a3, 0xffff
ori $a3, $a3, 0xffff    # $a3 = -1

#test add
add $s0, $a0, $a2   # ++
add $s1, $a0, $a3   # +-
add $s2, $a3, $a3   # --

# test sw
ori $t0, $0, 0x0000
sw  $a0, 0($t0)
sw  $a1, 4($t0)
sw  $a2, 8($t0)
sw  $a3, 12($t0)
sw  $s0, 16($t0)
sw  $s1, 20($t0)
sw  $s2, 24($t0)

# test lw
lw  $a0, 0($t0)
lw  $a1, 12($t0)
sw  $a0, 28($t0)
sw  $a1, 32($t0)
```

```
# test beq
ori $a0, $0, 1
ori $a1, $0, 2
ori $a2, $0, 1
beq $a0, $a1, ne
beq $a0, $a2, eq

ne:
sw  $a0, 36($t0)
eq:
sw  $a1, 40($t0)
```

4.

```
# test sll,j
ori     $t0, $0, 3
nop
test:
sll     $t0, $t0, 2
nop
sll     $t1, $t0, 2
nop
j       test
```

5.

```
.data
    my_stack: .space 200
.text
    ori $s0,4
    ori $s1,10
    ori $s2,1
    addu $a0,$0,$s1
    jal func
    jal end

func:
    sw $ra,0($t1)
    addu $t1,$t1,$s0

    beq $a0,$s2,func_ret
        sw $a0,0($t1)
        addu $t1,$t1,$s0

        subu $a0,$a0,$s2
        jal func

        subu $t1,$t1,$s0
        lw $a0,0($t1)
        addu $v0,$v0,$a0

        jal func_end

func_ret:
```

```
    addu $v0,$0,$s2
    j func_end

func_end:
    subu $t1,$t1,$s0
    lw $ra,0($t1)
    jr $ra
end:
    lui $t7,16
```

主要测试`jal,j,jr`等跳转指令

四、思考题

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？

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

DM内部将数据按字存储，每个字4字节32位，因此addr要与字对齐。

addr信号由ALU产生

2. 思考Verilog语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣。

① 门级描述：`assign ctrl=op[0]&op[1]$...`

② `assign`+三目运算符：`assign ctrl=op==0?1:0...`

③ 行为级描述：`case(op)`

优缺点：

① 设计复杂，易出错。

② 可以直接赋值，无需放到always块中。但不如case直观

③ 直观清晰，易于修改不易出错

3. 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所驱动的部件具有什么共同特点？

   DM，IM，GRF都需要清零。都有存储值，都有对应初始化的值。

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 。

对补码的运算便是通过直接的加法便能得到最终结果，所的结果的机器码是相同的，仅在译码时根据相应的指令赋予不同的含义。

5. 根据自己的设计说明单周期处理器的优缺点。

优点：1. 设计方便，结构简单

2. 可以没有延时槽

缺点：1. 单个周期时间很长，浪费时间资源

2. 同一时间所有元件只有一部分工作，浪费器件资源