计组 P3 设计文档

18231047 王肇凯

一、 模块规格

1、IFU

端口定义：

信号名	方向	描述
nPC_sel	I	当指令为需要跳转的指令时为 1，否则为 0
Zero	I	当 ALU 两个输入信号相等时，zero 为 1，否则为 0
Clk	I	时钟信号
Reset	I	复位信号
Offset[15:0]	I	跳转偏移量
DO[31:0]	O	从 PC 中取出的 32 位指令操作数

功能定义：

序号	功能名称	功能描述
1	计算 npc	当 nPC_sel 和 zero 同时为 1 时，PC=PC+4+offset*4 否则 PC=PC+4
2	取指令	根据当前 PC 值，从指令寄存器中取出 32 位操作数 （将 PC[6:2]与 IM[4:0]相连）
3	复位	当 Reset=1 时，设置为 0x00000000

2、GRF

端口定义：

信号名	方向	描述
A1[4:0]	I	指令的第 21 到第 25 位
A2[4:0]	I	指令的第 16 到第 20 位
A3[4:0]	I	当 RegDst 信号为 0 时，指令的第 16 到第 20 位； 当 RegDst 信号为 1 时，指令的第 11 到第 15 位；
RegWr	I	写使能信号，当为 1 时，将 WD 的值写入 A3 对应的寄存器
WD[31:0]	I	回写寄存器值，当 RegWr 为 1 时，WD 的值写入 A3
Clk	I	时钟信号
RD1[31:0]	O	从指令中读取的第一个操作数
RD2[31:0]	O	从指令中读取的第二个操作数

功能定义：

序号	功能名称	功能描述
1	读取操作数	根据当前 PC 值，读取第一位和第二位操作数
2	回写	当 RegWrite 信号为 1 时，在时钟上升沿，将 busW 的值写入 rt/rd 寄存器中

3、ALU

端口定义：

信号名	方向	描述
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
ALUctr[1:0]	I	当等于 00 时，ALU 执行加法运算； 当等于 01 时，ALU 执行减法运算； 当等于 10 时，ALU 执行或运算。
C[31:0]	O	将两个操作数进行运算后的结果
zero	O	当输入的两个操作数相等时，输出 1； 当输入的两个操作数不相等时，输出 0。

功能定义：

序号	功能名称	功能描述
1	算数逻辑运算	根据 ALUctr 的值，将 A 和 B 进行加法、减法、或运算
2	比较两个数是否相等	当 A 和 B 相等时，zero 信号为 1； 否则为 0。

4、DM

端口定义：

信号名	方向	描述
Ad[4:0]	I	C 的第 2 位到第 6 位数字，作为读写内存的地址
Din[31:0]	I	当写使能信号有效时，将 DIn 写入 Ad 所表示地址的内存
MemWrite	I	写使能信号。当为 1 时，将 Din 写入 Ad 所表示的地址
clk	I	时钟信号
reset	I	复位信号
DO[31:0]	O	Ad 地址的数据

功能定义：

序号	功能名称	功能描述
1	写入内存数据	当写使能信号有效时，将 DIn 写入 Ad 地址的内存
2	读取内存数据	将 Ad 所代表地址的数据输出
3	复位功能	当 reset 信号有效时，将 RAM 清零

5、EXT

端口定义：

信号名	方向	描述
Imm[15:0]	I	输入的立即数
EXTop[1:0]	I	当 EXTop 为 00 时，将立即数进行无符号扩展； 当 EXTop 为 01 时，将立即数进行有符号扩展； 当 EXTop 为 10 时，将立即数进行高位扩展；
EXTout[31:0]	O	扩展后的立即数输出

功能定义：

序号	功能名称	功能描述

1	无符号扩展	当 EXTop 为 00 时，将立即数进行无符号扩展并输出
2	有符号扩展	当 EXTop 为 01 时，将立即数进行有符号扩展并输出
3	高位扩展	当 EXTop 为 10 时，将立即数进行高位扩展并输出

二、控制器

真值表：

func	100001	100011						000000
op	000000	000000	001101	100011	101011	000100	001111	000000
	addu	subu	ori	lw	sw	beq	lui	nop
RegDst	1	1	0	0	x	x	0	x
ALUsrc	0	0	1	1	1	0	1	x
MemtoReg	0	0	0	1	x	x	0	x
RegWrite	1	1	1	1	0	0	1	0
MemWrite	0	0	0	0	1	0	0	0
nPC_sel	0	0	0	0	0	1	0	0
ExpOP[1:0]	x	x	00	01	01	x	10	x
ALUctr[1:0]	add	sub	ori	add	add	sub	add	x

RegDst	0：将 rt 寄存器的值输入到 GRF 的 A3 接口 1：将 rd 寄存器的值输入到 GRF 的 A3 接口
ALUsrc	0：将 GRF 的 B 接到 ALU 的 B 接口 1：将 EXT 的 EXTout 接到 ALU 的 B 接口
MemtoReg	0：将 ALU 的 C 接到 GRF 的 busW 接口 1：将 DM 的 Dataout 接到 RGF 的 busW 接口
RegWr	0：不对 GRF 进行写操作 1：将 GRF 的 WD 写入 A3 中
Br	0：该指令不是 beq 1：该指令时 beq
EXTop	00：将立即数进行无符号扩展 01：将立即数进行有符号扩展 10：将立即数进行高位扩展
OP	00：进行加法操作 01：进行减法操作 10：进行或操作

三、测试程序

```
#ori
ori $a0,$0,0x100
ori $a1,$a0,0x123
#lui
lui $a2,456
```

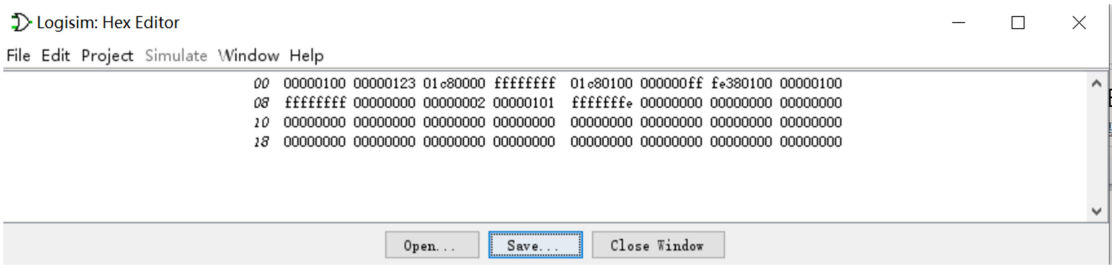
```
lui $a3,0xffff
ori $a3,$a3,0xffff
#addu
addu $s0,$a0,$a2
addu $s1,$a0,$a3
addu $s4,$a3,$a3
#subu
subu $s2,$a0,$a2
subu $s3,$a0,$a3
#sw
sw $a0,0($0)
sw $a1,4($0)
sw $a2,8($0)
sw $a3,12($0)
sw $s0,16($0)
sw $s1,20($0)
sw $s2,24($0)
sw $s3,44($0)
sw $s4,48($0)
#lw
lw $a0,0($0)
lw $a1,12($0)
sw $a0,28($0)
sw $a1,32($0)
#beq
ori $a0,$0,1
ori $a1,$0,2
ori $a2,$0,1
beq $a0,$a1,loop1
beq $a0,$a2,loop2

loop1: sw $a0,36($t0)
loop2: sw $a1,40($t0)
```

DM 测试预期：

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
0	0x00000100	0x00000123	0x01c80000	0xffffffff	0x01c80100	0x000000ff	0xfe380100	0x0000010(
32	0xffffffff	0x00000000	0x00000002	0x00000101	0xfffffffe	0x00000000	0x00000000	0x0000000(

DM 实际结果：



预期与实际一致，结果表明 CPU 通过了测试。

四、思考题

1. 若 PC（程序计数器）位数为 30 位，试分析其与 32 位 PC 的优劣

   优：占用的内存少，操作方便

   劣：没有 32 位 PC 存储的内容多

2. 现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用寄存器，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

   合理。ROM 只读不写，符合指令集的要求

   RAM 能读能写，符合数据寄存器的要求

   寄存器文件用 32 个具有写使能的 32 位寄存器，符合寄存器文件随时可写可读的要求。

3. 结合上文给出的样例真值表，给出 RegDst， ALUSrc， MemtoReg，RegWrite，nPC_Sel，ExtOp 与 op 和 func 有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

   RegDst

   =audu+subu

   =(~op0&~op1&~op2&~op3&~op4&~op5&func0&~func1&~func2&~func3&~func4&func5)+(~op0&~op1&~op2&~op3&~op4&~op5&func0&func1&~func2&~func3&~func4&func5)

   ALUSrc

   =ori+lw+sw+lui

   =(op0&~op1&op2&op3&~op4&~op5)+

(op0&op1&~op2&~op3&~op4&op5)+

(op0&op1&~op2&op3&~op4&op5)+

(op0&op1&op2&op3&~op4&~op5)

MemtoReg=lw=(op0&op1&~op2&~op3&~op4&op5)

RegWrite=addu+subu+ori+lw+lui

=(~op0&~op1&~op2&~op3&~op4&~op5&func0&~func1&~func2&~func3&~func4&func5)+(~op0&~op1&~op2&~op3&~op4&~op5&func0&func1&~func2&~func3&~func4&func5)+

(op0&~op1&op2&op3&~op4&~op5)+

(op0&op1&~op2&~op3&~op4&op5)+

(op0&op1&op2&op3&~op4&~op5)

MemWrite=sw=(op0&op1&~op2&op3&~op4&op5)

nPC_Sel=beq=(~op0&~op1&op2&~op3&~op4&~op5)

ExtOp0=lw+sw=(op0&op1&~op2&~op3&~op4&op5)+

(op0&op1&~op2&op3&~op4&op5)

ExtOp1=lui=(op0&op1&op2&op3&~op4&~op5)

ALUctr1=ori=(op0&~op1&op2&op3&~op4&~op5)

ALUctr2=subu+beq=(~op0&~op1&~op2&~op3&~op4&~op5&func0&func1&~func2&~func3&~func4&func5)+(~op0&~op1&op2&~op3&~op4&~op5)

4. 充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式， 请给出化简后的形式。

RegDst = ~op0

ALUSrc = op0

MemtoReg = op5

RegWrite = ~op3~op2 + op3~op1

nPC_sel = ~op3op2

ExtUp=op5

5. 事实上,实现 nop 空指令,我们并不需要将它加入控制信号真值表，

为什么？请给出你的理由。

当指令为 nop 时，所有的控制信号都为 0，不读不写，相当于什么都不执行。

在设计的控制信号中，当出现的信号不满足“addu, subu, ori, lw, sw, beq, lui, nop”其中之一时，所有的控制信号都为 0，不读不写，满足 nop 指令，所以，不需要将它加入真值表，一样可以实现 nop 功能。

6. 前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM 片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。

   如果是 j 指令等涉及到绝对地址的指令，就利用片选信号驱动 MUX 选择原有地址减去数据偏移量。

7. 除了编写程序进行测试外，还有一种验证 CPU 设计正确性的办法——形式验证。 **形式验证**的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)”了解相关内容后，简要阐述相比与测试，形式验证的优劣。

   优点：
   对指定描述的所有可能的情况进行验证；
   借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，不需要开发测试。

   缺点：
   随着 CPU 结构复杂度的增加，运算量呈指数型增长。