P4 报告

一. CPU 设计文档

1. 数据通路设计

A． IM

接口定义

接口名称	类型	描述
clk	I	时钟信号
clr	I	复位信号
branch	I	跳转信号，为 1 时有效
[31:0]npc	I	输入在跳转指令时需要跳转的地址
instr	O	输出取出的指令
pc4	O	输出 pc+4 的值

功能描述

序号	功能名称	功能描述
1	复位	当 clr 信号有效时 pc 被置为 0x00003000
2.	取指令	取出 pc 对应地址里的指令
3	生成下一指令地址	若 branch 有效，下一指令地址为 npc 否则为 pc+4

B. GPR

端口定义

接口名	类型	描述
clk	I	时钟信号
reset	I	异步复位信号，1 为有效
we	I	写使能信号，1 为有效
a1[4:0]	I	RD1 端口输出的寄存器的编号
a2[4:0]	I	RD2 端口输出的寄存器的编号
a3[4:0]	I	要写入的寄存器的编号
wd	I	要写入选定寄存器的数据
rd1	O	输出 A1 端口选中寄存器的值

rd2	O	输出 A2 端口选中寄存器的值

功能描述

序号	功能名称	功能描述
1	复位	reset 信号有效时，所有寄存器的值置为 0
2	读寄存器	RD1,RD2 输出 A1,A2 选中寄存器的值
3	写寄存器	we 信号有效时 A3 选中的寄存器的值被写为 WD 的值

C. EXT

端口定义

接口名	类型	描述
imm[15:0]	I	输入 16 位待扩展立即数
extout[31:0]	O	输出扩展后 32 位数据
extop[1:0]	I	功能选择 00 无符号扩展，01 有符号扩展，02Lui 扩展

功能说明

序号	功能名称	功能描述
1	无符号扩展	高位补 16 个 0
2	有符号扩展	高位补 16 个符号位
3	Lui 扩展	低位补 16 个 0

D:CMP

端口定义

接口名	类型	描述
[31:0]a	I	输入待比较数 a
[31:0]	I	输入待比较数 b
[31:0]zero32	O	输出 zeroextent(a==b)

功能说明

序号	功能名称	功能描述
1	相等判断	若 a==b 时有效

E: NPC

端口定义

接口名	类型	描述
npcop[1:0]	I	功能选择 00 依据 Imm 字段计算下一指令地址 01 依据指令后 26 位与 pc+4 高四位计算地址 02 输出 rd1 中的地址
pc4	I	输入 pc+4
extout	I	输入 imm 进行 16 位扩展后的信号
rd1	I	输入 gpr 中 a1 选定寄存器的值
intsr	I	输入当前正在执行的指令
npcout	O	输出计算出的下一指令

功能描述

序号	功能名称	功能描述
1	实现立即数跳转	依据 Imm 字段计算下一指令地址
2	实现 j 类型跳转	依据指令后 26 位与 pc+4 高四位计算地址
3	实现寄存器跳转	输出 rd1 中的地址

F: ALU

接口定义

接口名	类型	描述
a[31:0]	I	输入 alu 的第一个操作数
b[31:0]	I	输入 alu 的第二个操作数
aluop[1:0]	I	alu 功能选择 00 为与运算，01 为加运算，10 为减运算，11 为或运算，
aluout[31:0]	O	输出运算结果

功能描述

序号	功能名称	功能描述
1	加运算	输出 a1+a2
2	减运算	输出 a1-a2
3	或运算	输出 a1|a2

G: DM

接口描述

接口名	类型	描述
addr[31:0]	I	DM 输入的地址
wd[31:0]	I	DM 要写入的数据
clk	I	时钟信号
clr	I	异步复位信号
we	I	写使能信号,1 为有效,将 A 端口选中的字写为 INPUT 中数据
dmout[31:0]	O	输出 addr 端口选中的字的值

功能描述

序号	功能名称	功能描述
1	复位	clr 信号有效时，所有主存的值置为 0
2	读主存	输出 addr 端口选中的字的值
3	写主存	we 有效时将 addr 端口选中的字写为 wd 中数据

H:DATAPATH

接口描述

接口名	类型	描述
clk	I	时钟信号
clr	I	复位信号
branch	I	跳转信号
regwrite	I	寄存器写使能信号
memwrite	I	主存写使能信号
npcop[1:0]	I	跳转指令地址模式选择信号
extop[1:0]	I	扩展模式选择
aluop[1:0]	I	alu 运算模式选择
alusrc	I	alu 输入信号来源
regdst[1:0]	I	寄存器写入地址选择
memtoreg	I	寄存器写入数据选择

isbeq	I	是否为 beq 指令
instr[31:0]	I	输出当前指令

2. 控制器设计

	addu	subu	ori	lw	sw	beq	lui	jal	jr
branch	0	0	0	0	0	1	0	1	1
regwrite	1	1	1	1	0	0	1	1	0
memwrite	0	0	0	0	1	0	0	0	0
alusrc	0	0	1	1	1	0	0	0	0
npcop	0	0	0	0	0	0	0	1	2
extop	0	0	0	1	1	1	2	0	0
aluop	1	2	3	1	1	0	0	0	0
regdst	0	0	1	1	1	0	1	2	0
memtoreg	0	0	0	1	1	0	2	3	0

3. 测试程序

```
lui $s0,0x1234
ori $s0,0x5678
sw $s0,0($0)
lw $s1,0($0)
lhu $s2,0($0)
lhu $s3,2($0)
lbu $s4,0($0)
lbu $s5,1($0)
lbu $s6,2($0)
lbu $s7,3($0)
sb $s7,4($0)
sb $s6,5($0)
sb $s5,6($0)
sb $s4,7($0)
sh $s3,8($0)
sh $s2,10($0)
```

测试结果预期：

```
@00003000: $16 <= 12340000
@00003004: $16 <= 12345678
@00003008: *00000000 <= 12345678
@0000300c: $17 <= 12345678
@00003010: $18 <= 00005678
@00003014: $19 <= 00001234
@00003018: $20 <= 00000078
@0000301c: $21 <= 00000056
@00003020: $22 <= 00000034
@00003024: $23 <= 00000012
@00003028: *00000004 <= 00000012
```

```
@0000302c: *00000004 <= 00003412
@00003030: *00000004 <= 00563412
@00003034: *00000004 <= 78563412
@00003038: *00000008 <= 00001234
@0000303c: *00000008 <= 56781234
```

```
lui $s0,0xffff
ori $s0,0xffff
ori $s1,1
slt $t7,$s0,$s1
beq $t7,$0,tar1
ori $t0,1
tar1:
slt $t7,$s1,$s0
beq $t7,$0,tar2
ori $t1,1
tar3:
jr $ra
tar2:
jal tar3
ori $t2,1tar2:
```

```
@00003000: $16 <= ffff0000
@00003004: $16 <= ffffffff
@00003008: $17 <= 00000001
@0000300c: $15 <= 00000001
@00003014: $ 8 <= 00000001
@00003018: $15 <= 00000000
@00003028: $31 <= 0000302c
@0000302c: $10 <= 00000001
```

二、思考题

L0.T2.1：

addr 是[11:2]和[9:0]在语法上完全等价，但实际上 addr 是 alu 运算结果 aluout 的第 11 到 2 位，所以用[11:2]更好

L0.T2.2

对 pc 寄存器进行清零 因为 pc 初始值必须是 0x00003000

对 gpr 中的 1-31 号寄存器进行清零 因为这些寄存器初始值为 0

对主存进行清零，因为主存的初始值为 0

L0.T4.1

第一种：if(op==6'b000011)begin

end

第二种：assign jal=~op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];

assign npcop[0]=jal&j;

第三种： `define jal op==6'b000011

L0.T4.2

第一种：优点：写着省事

缺点：不容易 debug 可读性奇差无比(其实除了省事全是缺点)

第二种：优点：输出信号不需要用寄存器了，也可以直接观察某指令类型是否被命中

缺点：写着太长了

第三种：优点：简洁明了，写着方便

缺点：要是用了 if else 得用寄存器而且还不容易确定分支语句执行了哪个分支，要是用 assign，不容易判断各个指令对应信号的值

L0.T5.1

add 执行的操作是在两个待操作的数前各自添加一个信号位，然后将其相加，与不添加信号位的相加结果比较，这个结果其实就是原来两个操作数直接相加的结果，若两个结果不同则发出异常，否则将相加结果存入待写寄存器

addu 的操作则是将两个待操作数直接相加，存入待写寄存器

若 add 不报异常，则两个结果自然完全相同

addiu 与 addi 同理。

L0.T5.2

单周期的优点：结构简单，复杂度低，每个周期只执行一条指令，不需要阻塞，不需要转发

缺点：时钟周期受到关键路径的制约，无法进一步缩小；每个周期只能同时执行

一条指令，使用的是最长的时钟周期，不能同时执行多条指令

L0.T5.3

通常 jal 向 31 号寄存器写入的下一指令地址会被保存在堆栈中，jr 所跳转的地址经常是从堆栈中取出