P4 Document

设计草稿

数据通路

见附图

reset控制同步复位。

PC

端口	方向	位宽
clk	I	1
reset	I	1
in	I	32
out	O	32

注：初始地址为0x00003000。

NPC

端口	方向	位宽
in	I	32
Imm26	I	26
RA	I	32
NPCOp	I	2
zero	I	1
PC4	O	32
out	O	32

NPCOp	NPC
00	PC+4
01	PC+4+sign_extend(offset||00)
10	PC31..28||instr_index||00
11	GPR[rs]

IM

端口	方向	位宽
in	I	32
instr	O	32
注：容量为32x4096。		

> 取in[11:2]

GRF

端口	方向	位宽
clk	I	1
reset	I	1
RegWrite	I	1
a1	I	5
a2	I	5
a3	I	5
WD	I	32
pc	I	32
RD1	O	32
RD2	O	32
注：容量为32x32。		
0号寄存器不读入		

```
$display("@%h: $%d <= %h", pc, a3, WD);
```

CTRL

```
input wire [5:0] op;
input wire [5:0] func;
reg [13:0] tmp;
```

指令	RegWrite	RegDst	ALUSrc	MemWrite	WDSrc	NPCOp	ALUOp	EXTOp
add(000000+100000)	1	01	0	0	00	00	000	00
sub(000000+100010)	1	01	0	0	00	00	001	00
ori(001101)	1	00	1	0	00	00	010	00
lw(100011)	1	00	1	0	01	00	000	01
sw(101011)	0	00	1	1	00	00	000	01
beq(000100)	0	00	0	0	00	01	001	00
lui(001111)	1	00	1	0	00	00	000	10
jal(000011)	1	10	0	0	10	10	000	00
jr(000000+001000)	0	00	0	0	00	11	000	00

注：`add`与`sub`实际上为`addu`与`subu`，不考虑溢出情况。

RegDst

[4:0] mux1=

> 00: GPR[rt]
> 01: GPR[rd]
> 10: GPR[31]

ALUSrc

[31:0] mux2=

> 0: RD2
> 1: EXTOut

WDSrc

[31:0] mux3=

> 00: ALUOut
> 01: MemOut
> 10: PC4

ALU

端口	方向	位宽
a	I	32
b	I	32
ALUOp	I	3
ALUResult	O	32
zero	O	1

ALUOp	运算
000	加
001	减
010	或

EXT

端口	方向	位宽
Imm16	I	16
EXTOp	I	2
EXTResult	O	32

EXTOp	扩展
00	零扩展
01	符号扩展
10	加载至高位

DM

端口	方向	位宽
in	I	32
addr	I	32
MemWrite	I	1
clk	I	1
reset	I	1
pc	I	32
out	O	32
注：容量为32x3072。		
取addr[13:2]		

`$display("@%h: *%h <= %h", pc, addr, in);`

测试方案

p4弱测使用的测试指令：

```
.text
ori $gp, $zero, 0
ori $sp, $zero, 0
ori $at, $zero, 0x3456
add $at, $at, $at
lw  $at, 4($zero)
sw  $at, 4($zero)
lui $v0, 0x7878
sub $v1, $v0, $at
lui $a1, 0x1234
ori $a0, $zero, 5
nop
sw  $a1, -1($a0)
lw  $v1, -1($a0)
beq $v1, $a1, a
#never do:
nop
j b
nop
```

```
a:
ori $a3, $v1, 0x404
beq $a3, $v1, b #no jump
nop
lui $t0, 0x7777
ori $t0, $t0, 0xffff
sub $zero, $zero, $t0
ori $zero, $zero, 0x1100
add $t2, $a3, $a2
ori $t0, $zero, 0
ori $t1, $zero, 1
ori $t2, $zero, 1
c:
add $t0, $t0, $t2
beq $t0, $t1, c

jal b
nop
add $t2, $t2, $t2
d:
j d #end in 0x3084

b:
add $t2, $t2, $t2
jr  $ra
nop
```

机器码为：

```
341c0000
341d0000
34013456
00210820
8c010004
ac010004
3c027878
00411822
3c051234
34040005
00000000
ac85ffff
8c83ffff
10650003
00000000
10000011
00000000
```

```
34670404
10e3000e
00000000
3c087777
3508ffff
00080022
34001100
00e65020
34080000
34090001
340a0001
010a4020
1109fffe
0c000c22
00000000
014a5020
1000ffff
014a5020
03e00008
00000000
```

ISE正确输出为:

```
@00003000: $28 <= 00000000
@00003004: $29 <= 00000000
@00003008: $ 1 <= 00003456
@0000300c: $ 1 <= 000068ac
@00003010: $ 1 <= 00000000
@00003014: *00000004 <= 00000000
@00003018: $ 2 <= 78780000
@0000301c: $ 3 <= 78780000
@00003020: $ 5 <= 12340000
@00003024: $ 4 <= 00000005
@0000302c: *00000004 <= 12340000
@00003030: $ 3 <= 12340000
@00003044: $ 7 <= 12340404
@00003050: $ 8 <= 77770000
@00003054: $ 8 <= 7777ffff
@00003060: $10 <= 12340404
@00003064: $ 8 <= 00000000
@00003068: $ 9 <= 00000001
@0000306c: $10 <= 00000001
@00003070: $ 8 <= 00000001
@00003070: $ 8 <= 00000002
@00003078: $31 <= 0000307c
@00003088: $10 <= 00000002
@00003080: $10 <= 00000004
```

思考题

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] ？

> 信号addr来自于ALUOut，`addr = ALUOut[11:2]`。
> 容量1024=2的10次方，因此地址信号addr的位宽为10位。DM是按字寻址的，1字=4字节，将[9:0]左移2位即为[11:2]。

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。

> 指令对应的控制信号如何取值:

```
case (op)
    `ORI: begin
        tmp <= 13'b1_00_1_0_00_00_010_00;
    end
    ...
endcase
assign {RegWrite, RegDst, ALUSrc, MemWrite, WDSrc, NPCOp, ALUOp,
EXTOp} = tmp;
```

> 控制信号每种取值所对应的指令:

```
case (op)
    `ORI: begin
        ori = 1;
    end
    ...
endcase
assign RegWrite = add + sub + ori + lw + lui + jal;
...
```

> 个人倾向于第一种译码方式，它能很容易从控制信号真值表转化得到，并且层次分明，易于增加指令和检查。

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：在clk上升沿检查reset，若reset有效，复位。
> `always @(posedge clk) if(reset)...`
> 异步复位：在任意时刻检查reset，reset有效时立即复位，与clk无关。
> `always @(posedge clk or posedge reset) if(reset)...`

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。

> 若忽略溢出情况， `add` 指令具体操作中的 `GPR[rd] = temp31...0 = GPR[rs] + GPR[rt]` ，即等价于 `addu指令` 。