思考题

一. 本设计为求准确，转发部分采取傻瓜式转发

（能转就转不考虑,转没转过,反正转发又不是 STALL 只要不转发错了也不降低性能，全力转发去呗，总比写出 bug 再挂一个课上测试强 QAQ……求助教霸霸放过我这只垂死挣扎的小辣鸡）

共分为 3 类：

a) 处理方法为直接 STALL 一周期的：

1. D 级 rs 或 rt 的 tuse=0,E 级指令由 ALU 写入相同的寄存器（不是 $0 寄存器）

   示例：ori $x,$0,1　beq $x,$y,gxp

   或 ori $x,$0,0x3000　jr $x

2. D 级 rs 或 rt 的 tuse=0,E 级指令由 DM 写入相同寄存器（不是$0 寄存器）

   示例 lw $x,z($y)　beq $x,$y,gxp

   或 lw $x,z($y)　jr $x

3. D 级 rs 或 rt 的 tuse=1,E 级指令由 DM 写入相同寄存器（不是$0 寄存器）

   示例 lw $x,z($y)　addu $x,$x,$x

4. D 级 rs 或 rt 的 tuse=0,M 级指令由 DM 写入相同寄存器（不是$0 寄存器）

   示例：lw $x,z($y)　xxxxxx　beq $x,$y,gxp

   或 lw $x,z($y)　xxxxxx　jr $x

b) 处理方法为外部显式转发的（下述情况均已排除读写零号寄存器的坑，不再反复说明）

/*----------D 级转发 MUX 构成----------*/

1. E 级指令由 PC8 写某寄存器且 D 级读该寄存器的：从 E 级将 PC8

向 D 级转发

示例 Jal xxxxxxxx　addu $ra,$ra,$x

2. E 级指令由 EXT 写某寄存器且 D 级读该寄存器的：从 E 级将 EXT 向 D 级转发

   示例 lui $x,0xffff　beq $x,$0,gxp

3. M 级指令由 ALU 写某寄存器且 D 级读该寄存器的：从 M 级将 AO 向 D 级转发

   示例 ori $x,0x3000　nop　addu $x,$x,$x

4. M 级指令由 PC8 写某寄存器且 D 级读该寄存器的：从 M 级将 PC8 向 D 级转发

   示例 jal gxp　nop　gxp:addu $ra,$ra,$x

/*------------------E 级转发 MUX 构成--------------------*/

5. M 级指令由 PC8 写入某寄存器且 E 级读该寄存器的：从 M 级将 PC8 转发至 E 级

   示例 Jal xxxxxxxx　addu $ra,$ra,$x

6. M 级指令由 ALU 写入某寄存器且 E 级读该寄存器的：从 M 级将 AO 转发至 E 级

   示例 addu $x,$x,$x　　addu $x,$x,$x

7. W 级指令写入某寄存器且 E 级读该寄存器的：从 W 级将寄存器堆输入数据转发至 E 级

   示例：addu $x,$y,$z xxxxxxx　addu $x,$x,$x

/*----------M 级转发 MUX 构成----------*/

8. W 级指令写入某寄存器且 M 级读该寄存器的：从 W 级将寄存器堆输入数据转发至 M 级

   示例：addu $x,$x,$x　sw $x,0($y)

c) 通过寄存器堆内部支撑转发的

W 级指令写入某寄存器且 D 级读该寄存器的

示例：addu $x,$y,$z xxxxxxx　yyyyyyyyy addu $x,$x,$x

cpu 设计文档

一. 数据通路设计

1. IFU

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
npcout[31:0]	I	输入在跳转指令时需要跳转的地址
stall	I	暂停信号
instr	O	输出取出的指令
pc8	O	输出 pc+8 的值

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
4	stall 响应	若 stall 有效，锁定 pc 寄存器使其值不再改变

2. grf

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(进行转发)
rd2	O	输出 A2 端口选中寄存器的值（进行转发）

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

3. npc

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

4. ext

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

5. cmp

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

6. alu

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 输出 b，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
4	输出 b	为提高 lui 等指令的性能并降低复杂度而预留

7. dm

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

8. 各级流水线寄存器

```
module Dpipe (clk,clr,en,
              instr_i,pc8_i,
              instr_d,pc8_d);

module Epipe(clk,clr,
res_d,instr_d,a1_d,a2_d,a3_d,v1_d,v2_d,oth_d,pc8_d,
res_e,instr_e,a1_e,a2_e,a3_e,v1_e,v2_e,oth_e,pc8_e);

module Mpipe(clk,clr,
res_e,instr_e,a2_e,a3_e,v2_e,ao_e,pc8_e,
res_m,instr_m,a2_m,a3_m,v2_m,ao_m,pc8_m);

module Wpipe(clk,clr,
res_m,instr_m,a3_m,ao_m,dr_m,pc8_m,
res_w,instr_w,a3_w,ao_w,dr_w,pc8_w);
```

9. datapath

接口名	类型	描述
clk	I	时钟信号
clr	I	复位信号
stall	I	暂停信号

abus[39:0]	O	各级使用的寄存器编号的集合
		assign abus[4:0]=a1_d;
		assign abus[9:5]=a2_d;
		assign abus[14:10]=a1_e;
		assign abus[19:15]=a2_e;
		assign abus[24:20]=a2_m;
		assign abus[29:25]=a3_e;
		assign abus[34:30]=a3_m;
		assign abus[39:35]=a3_w;

resbus[8:0]	O	各级 res 寄存器值
		assign resbus[2:0]=res_e;
		assign resbus[5:3]=res_m;
		assign resbus[8:6]=res_w;

forwardbus[12:0]	I	各个转发 MUX 控制信号的值
		assign fv1dctrl=forwardbus[2:0];
		assign fv2dctrl=forwardbus[5:3];
		assign faluaectrl=forwardbus[8:6];
		assign falubectrl=forwardbus[11:9];
		assign fdmmctrl=forwardbus[12];

res_d[2:0]	I	写入 res_d 的值
branch	I	跳转信号
npcop[1:0]	I	npc 控制信号
extop[1:0]	I	ext 控制信号

regdst[1:0]	I	控制写入寄存器选择
otherctrl[1:0]	I	控制 D 级其他写入部件的选择
aluop[2:0]	I	控制 alu 工作信号
alusrc[1:0]	I	控制 alu b 输入来源
memwrite	I	主存写使能
memtoreg	I	控制写寄存器信号来源
regwrite	I	寄存器写使能
isbeq	I	是 beq 指令
instr_d[31:0]	O	向上层各控制器输出各级指令备用
instr_e[31:0]		
instr_m[31:0]		
instr_w[31:0]		

二．控制器设计

1.常规控制器设计

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

2.AT 控制器

指令	tuse_rs	tuse_rt	res

addu	1	1	alu
subu	1	1	alu
ori	1		alu
lw	1		dm
sw	1	2	nw
beq	0	0	nw
jr	0		nw
jal			pc
j			
lui			ext

3.STALL 控制器

```
assign stall_rs01e=(tuse_rs==0)&&(res_e==`alu)&&(a1_d==a3_e)&&(a1_d!=0);
assign stall_rs02e=(tuse_rs==0)&&(res_e==`dm)&&(a1_d==a3_e)&&(a1_d!=0);
assign stall_rs12e=(tuse_rs==1)&&(res_e==`dm)&&(a1_d==a3_e)&&(a1_d!=0);
assign stall_rs01m=(tuse_rs==0)&&(res_m==`dm)&&(a1_d==a3_m)&&(a1_d!=0);
assign stall_rs=stall_rs01e|stall_rs02e|stall_rs12e|stall_rs01m;

assign stall_rt01e=(tuse_rt==0)&&(res_e==`alu)&&(a2_d==a3_e)&&(a2_d!=0);
assign stall_rt02e=(tuse_rt==0)&&(res_e==`dm)&&(a2_d==a3_e)&&(a2_d!=0);
assign stall_rt12e=(tuse_rt==1)&&(res_e==`dm)&&(a2_d==a3_e)&&(a2_d!=0);
assign stall_rt01m=(tuse_rt==0)&&(res_m==`dm)&&(a2_d==a3_m)&&(a2_d!=0);
assign stall_rt=stall_rt01e|stall_rt02e|stall_rt12e|stall_rt01m;

assign stall=stall_rs|stall_rt;
```

4 转发控制器

```
31  assign fv1dctrl=(res_e==`pc && a1_d==a3_e && a1_d!=0)?`fd_pc8_e:
32                  (res_e==`other && a1_d==a3_e &&a1_d!=0)?`fd_oth_e:
33                  (res_m==`alu && a1_d==a3_m && a1_d!=0)?`fd_ao_m:
34                  (res_m==`pc && a1_d==a3_m && a1_d!=0)?`fd_pc8_m:
35                  `fd_rd;
36  assign fv2dctrl=(res_e==`pc && a2_d==a3_e && a2_d!=0)?`fd_pc8_e:
37                  (res_e==`other && a2_d==a3_e &&a2_d!=0)?`fd_oth_e:
38                  (res_m==`alu && a2_d==a3_m && a2_d!=0)?`fd_ao_m:
39                  (res_m==`pc && a2_d==a3_m && a2_d!=0)?`fd_pc8_m:
40                  `fd_rd;
41  assign faluaectrl=(res_m==`pc &&a1_e==a3_m&&a1_e!=0)?`falue_pc8_m:
42                    (res_m==`alu &&a1_e==a3_m&&a1_e!=0)?`falue_ao_m:
43                    (res_w!=`nw && a1_e==a3_w&&a1_e!=0)?`falue_wd_w:
44                    `falue_v;
45  assign falubectrl=(res_m==`pc &&a2_e==a3_m&&a2_e!=0)?`falue_pc8_m:
46                    (res_m==`alu &&a2_e==a3_m&&a2_e!=0)?`falue_ao_m:
47                    (res_w!=`nw && a2_e==a3_w&&a2_e!=0)?`falue_wd_w:
48                    `falue_v;
49  assign fdmmctrl=(res_w!=`nw && a2_m==a3_w &&a2_m!=0)?1:0;
```

测试程序

1. 暂停测试：

ori $s0,1
beq $s0,$0,tar:
nop
nop
tar:nop
测试结果:ISIM 中观察到 stall 信号置 1 的现象
ori $a0,1
sw $a0,0($0)
lw $s0,0($0)
beq $s0,$0,tar:
nop
nop
tar:nop
测试结果：ISIM 中监视到两次 STALL
ori $a0,1
sw $a0,0($0)
lw $s0,0($0)
addu $s0,$s0,$s0
测试结果：ISIM 中监视到一次 STALL

2. 测试

(本测试不依靠输出结果判断 CPU 是否正常工作，故未附上测试输出结果

进行测试时在 isim 中逐条指令运行，在需要观察转发控制器的转发情况的指

令后均有注释标注，观察控制器转发信号是否正确即可都达到测试目的)

ori $s0,$0,0x300c

```
jal $s0
addu $s0,$s0,$s0#在执行至此处时观察转发 mux 控制信号和转发情况
lui $s0,0xffff
beq $s0,$0,tar#在此处观察转发 mux 控制信号和转发情况
nop
tar:nop
lui $s0,0x0000
ori $s0,0x3000
nop
addu $s0,$s0,$s0#在此处观察 mux 控制信号和转发情况
jal gxp
nop
gxp:addu $ra,$ra,$ra#在此处观察转发 mux 控制信号
jal gxp2
nop
gxp2:addu $s0,$s0,$s0#在执行至此处时观察转发 mux 控制信号和转发情况
ori $a0,1
addu $a0,$a0,$a0
addu $a0,$a0,$a0#在此处观察 mux 控制信号
ori $a0,1
addu $a0,$a0,$a0
nop
addu $a0,$a0,$a0#在此处观察 mux 控制信号
sw $a0,0($0)#在此处观察 mux 控制信号
nop
addu $a0,$a0,$a0#在此观察控制信号
```