CPU 设计文档

一、 模块规格

1. GRF
   寄存器堆

A 端口定义

信号名	方向	描述
rs[4:0]	I	指令 21-25 位
rt[4:0]	I	指令 16-20 位
rd[4:0]	I	指令 11-15 位
pc[31:0]	I	输入当前的 pc 的值
alu_result[31:0]	I	输入 ALU 的运算结果
pc_4add[31:0]	I	输入 pc＋4 的值
DM_out[31:0]	I	输入从 DM 中输出的 32 位数据
Clk	I	时钟信号
Clr	I	清零信号，将32个寄存器中的值全部清零，高电平有效
RegWrite	I	寄存器写使能信号，高电平有效
RegDst	I	写入地址选择信号，为 0 时选择 rt，为 1 时选择 rd
jal_slt		写入地址与数据选择信号，若为 1 时选择 31，且数据选择 pc_4add
MemtoReg	I	写入数据选择信号，为 0 时选择 alu_result，为 1 时选择 DM_out
RD1	O	输出rs指定寄存器的32位数据
RD2	O	输出rt指定寄存器的32位数据

B 功能描述

序号	功能名称	描述
1	复位	当clk 上 升 沿 ，clr信号有效时，清空存储器的值
2	读数据	读A1(rs)，A2(rt)对应寄存器的值到RD1，RD2
3	写数据	当写使能信号RegWrite有效时，将选择后的RegData的值写入 选 择 后 RegAddr对应的寄存器中

2. ALU

算术逻辑部件运算器

A 端口定义

信号名	方向	描述
rs_out [31:0]	I	ALU的第一个备选运算数
rt_out [31:0]	I	ALU的第二个备选运算数
imm_ext[31:0]	I	ALU的第三个备选运算数
sa[4:0]	I	ALU的第三个运算数，用于特殊指令当中的位移
sll_slt	I	运算数选择，1：第一个运算数 A 选择 rt_out，1：第一个运算数 A 选择 rs_out
ALUSrc	I	运算数选择，1：第二个运算数 B 选择 imm_ext，1：第二个运算数 B 选择 rt_out
ALUO[2:0]	I	ALU信号控制器。控制ALU做什么运算
Result[31:0]	O	ALU的运算结果
Ztag	O	零标志位，为0时代表运算结果不为0，为1时代表运算结果为0

B 功能描述

序号	功能名称	描述
1 ALUO[2:0]	000 加	Result=A+B
	001 减	Result=A-B
	010 与	Result=A&&B
	011 或	Result=A|B
	100 非	Result=~A
	101 逻辑左移	Result=A<<sa
2	示零	当result=0时，零标志位为1，当result！=0时，零标志位为0

3. PC

A 端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	CP复位信号
Immediate[15:0]	I	输入的16位立即数
branch	I	是否进行branch信号，高电平有效
Ztag	I	零标志位
add26[25:0]	I	26位立即数
Jump	I	J指令信号，高电平有效
jr_alures	I	输入 jr 指令要跳转的 PC 值
jr_sel	I	是否要进行 jr 跳转
pc[31:0]	O	输出32位当前 pc
pc_4add[31:0]	O	输出32位当前 pc+4

B 功能描述

序号	功能名称	描述
1	PC自增 4	在每个时钟上升沿到达时，若reset、PCsel、jump信号均无效，PC＝PC+4
2	Branch if equal	在时钟上升沿到达时，若PCsel与ztag同时有效,且reset信号无效，PC=PC+4+sign_extend(immediate||0^2)
3	Jump	在时钟上升沿到达时，若jump信号有效,且reset信号无效，PC=PC31..28||26add||0^2
4	jr	在时钟上升沿到达时，若jr_sel信号有效，且reset信号无效，则 pc=jr_alures
5	同 步复位	当clk 上升沿，clr信号有效时，cp=0x00003000

4. IM

指令存储器

A 端口定义

信号名	方向	描述
pc[31:0]	I	输入32位pc 值
add26[25:0]	O	机器码0-25位
Immediate[15:0]	O	机器码0-15位
Func[5:0]	O	机器码位0-5位
Sa[5:0]	O	机器码6-10位
Rd[5:0]	O	机器码11-15位
Rt[5:0]	O	机器码16-20位
Rs[5:0]	O	机器码21-25位
Ocode[5:0]	O	机器码26-31位

B 功能描述

序号	功能名称	描述
1	取指令	取出 IM 中 pc 所指地址的指令
2	分割指令	将取出的指令进行分割，变为输出中的各个字段

5. EXT

16 位立即数扩展器

A 端口定义

信号名	方向	描述
Imm[15:0]	I	16位立即数
EXTslt[1:0]	I	扩展器功能选择信号
Imm_ext[31:0]	O	输出拓展后的32位数

B 功能描述

序号	功能名称	描述
EXTslt[1:0]	00 无符号	对立即数进行无符号拓展
	01 符号	对立即数进行符号拓展
	10 加载到高位	将立即数加载到高16位

6. DM

数据存储器

A 端口定义

信号名	方向	描述
Address[31:0]	I	输入32位地址
Data in[31:0]	I	要写入数据存储器的32位数据
pc[31:0]	I	输入当前的 pc 值
MemWrite	I	数据存储器写使能，高电平有效
Clk	I	时钟信号
Clr	I	复位信号，高电平有效
Data out[31:0]	O	输出32位数据

B 功能描述

序号	功能名称	描述
1	写数据	在每个时钟上升沿到达时，若MemWrite有效，将data in中的数据写入到address11-6位对应的存储区域
2	读数据	在时钟上升沿到达时，将address2-11位对应的存储区域的数据读出到data out
3	同步复位	当clk 上升沿，clr信号有效时，清空存储器的值

7.信号控制器

A 端口定义

信号名	方向	描述
Ocode[5:0]	I	六位指令操作码输入
Func[5:0]	I	六位指令函数码输入
Jump	O	是否执行j指令
RegDst	O	决定运算结果写入rt还是rd，若为0，写入rt，若为1，写入rd
ALUSrc	O	决定ALU的第二个操作数，若为0，则为rt，若为1，则为立即数
MemtoReg	O	决定要写入GRF的数据，若为0，写入ALU的运算结果，若为1，写入DM中的输出数据
RegWrite	O	寄存器堆写使能信号，高电平有效
MemWrite	O	数据存储器写使能信号，高电平有效
PC_sel	O	决定PC是否进行branch跳转，高电平有效
EXTo[1:0]	O	决定EXT进行何种拓展
ALUo[2:0]	O	决定ALU进行何种运算
Sll_slt	O	选择ALU的第一个操作数，若为0，为rs，若为1，为rt
jr_slt	O	是否执行 jr 指令的选择信号
jal_alt	O	是否执行 jal 指令的选择信号

B 各指令的控制信号

指令	Jump	RegDst	ALUSrc	MemtoReg	RegWrite	MemWrite	PC_sel	EXTo	ALUo	Sll_slt	jr_slt	jal_slt
Addu	0	1	0	0	1	0	0	00	000	0	0	0
Subu	0	1	0	0	1	0	0	00	001	0	0	0
Sll	0	1	0	0	1	0	0	00	101	1	0	0
Ori	0	0	1	0	1	0	0	00	011	0	0	0
Lui	0	0	1	0	1	0	0	10	000	0	0	0
Beq	0	0	0	0	0	0	1	00	001	0	0	0
Lw	0	0	1	1	1	0	0	01	000	0	0	0
Sw	0	0	1	0	0	1	0	01	000	0	0	0
J	1	0	0	0	0	0	0	00	000	0	0	0
jal	1	0	0	0	1	0	0	00	000	0	0	1
jr	0	0	0	0	0	0	0	00	000	0	1	0

8.暂停控制器

A 暂停表

D（IF / ID）当前指令　　E（ID / EX）Tnew　　M（EX / MEM）Tnew

指令类型	源寄存器	Tuse	cal_r 1/rd	cal_i 1/rt	load 2/rt	load 1/rt
beq	rs/rt	0				
cal_r	rs/rt	1				
cal_i	rs	1				
load	rs	1				
store	rs	1				
jr	rs	0				

B 输入输出

信号名	方向	描述
IR_D[31:0]	I	D 级指令
IR_E[31:0]	I	E 级指令
IR_M[31:0]	I	M 级指令
stall	O	输出暂停信号

C 功能

根据输入的指令判断各个级别的指令类型，需求的寄存器与要写入的寄存器，根据暂停表中的数据分析，判断其是否需要暂停，输出暂停信号。

9.转发控制器

A 转发部件设计

D 级：

RS_MUX

信号名	方向	描述
RD1[31:0]	I	grf rs 数据
E_Froward[31:0]	I	E 级回写数据
M_Froward[31:0]	I	M 级回写数据
selet[1:0]	I	选择信号
rs_final[31:0]	O	最终数据

RT_MUX

信号名	方向	描述
RD2[31:0]	I	grf rt 数据
E_Froward[31:0]	I	E 级回写数据
M_Froward[31:0]	I	M 级回写数据
selet[1:0]	I	选择信号
rt_final[31:0]	O	最终数据

ALU1_MUX

信号名	方向	描述
E_reg_rs[31:0]	I	E 级流水输出 rs
W_Froward[31:0]	I	W 级回写数据
M_Froward[31:0]	I	M 级回写数据
selet[1:0]	I	选择信号
alu1_final[31:0]	O	最终数据

ALU2_MUX

信号名	方向	描述
E_reg_rt[31:0]	I	E 级流水输出 rt
W_Froward[31:0]	I	W 级回写数据
M_Froward[31:0]	I	M 级回写数据
selet[1:0]	I	选择信号
alu2_final[31:0]	O	最终数据

WD_MUX

信号名	方向	描述
W_Froward[31:0]	I	W 级回写数据
M_Froward[31:0]	I	M 级回写数据
selet	I	选择信号
wd_final[31:0]	O	最终数据

B 转发控制单元

信号名	方向	描述
addr_E[4:0]	I	E 级回写地址
addr_M[4:0]	I	M 级回写地址
addr_W[4:0]	I	W 级回写地址
IR_D[31:0]	I	D 级指令
ID_E[31:0]	I	E 级指令
IR_M[31:0]	I	M 级指令
sel_D_RS[1:0]	O	D rs 选择信号
sel_D_RT[1:0]	O	D rt 选择信号
sel_E_ALU1[1:0]	O	E alu1 选择信号
sel_E_ALU1[1:0]	O	E alu2 选择信号
sel_M_WD	O	M wd 选择信号

二、 数据通路设计

部件	输入	LW
PC		
ADD4		PC
IM		PC
PC		ADD4
IR@D		IM
PC4@D		
RF	A1	IR@D[rs]
	A2	
EXT		IR@D[i16]
NPC	PC4	
	I26	
PC		
IR@E		IR@D
PC4@E		
RS@E		RF.RD1
RT@E		
EXT@E		EXT
ALU	A	RS@E
	B	EXT@E
IR@M		IR@E
PC4@M		
AO@M		ALU
RT@M		
DM	A	AO@M
	WD	
IR@W		IR@M
PC4@W		
AO@W		
DR@W		DM
RF	A3	IR@W[rt]
	WD	DR@W

需要定义的线网型变量如下：

```
wire [31:0] ins;
//im 输出指令
wire jump;
wire RegDst;
wire ALUSrc;
wire MemtoReg;
wire RegWrite;
wire MemWrite;
wire branch;
wire [1:0] extop;
wire [2:0] aluop;
wire sll_slt;
wire jr_slt;
wire jal_slt;
//总控信号
wire [31:0] alu_res;
wire Ztag;
//ALU 输出
wire [31:0]pc;
wire [31:0]pc_4add;
//PC 输出
wire [31:0]imm_ext;
//EXT 输出
wire [31:0] grfo_rs;
wire [31:0] grfo_rt;
//GRF 输出
wire [31:0] DM_out;
//DM 输出
wire change;
wire [31:0] npc;
//NPC 输出
wire [31:0]IR_D_out;
wire [31:0]PC4_D_out;
wire [4:0]Forward_Addr_D_out;
wire [31:0]Forward_Data_D_out;

//D 级流水寄存器输出
wire cmp_res;
//CMP 比较单元输出
wire [31:0]IR_E_out;
wire [31:0]PC4_E_out;
```

```
wire [31:0]RS_E_out;
wire [31:0]RT_E_out;
wire [31:0]EXT_E_out;
wire [4:0]Forward_Addr_E_out;
wire [31:0]Forward_Data_E_out;

//E 级别流水寄存器输出
wire [31:0]IR_M_out;
wire [31:0]PC4_M_out;
wire [31:0]AO_M_out;
wire [31:0]RT_M_out;
wire [4:0]Forward_Addr_M_out;
wire [31:0]Forward_Data_M_out;

//M 级流水寄存器输出
wire [31:0]IR_W_out;
wire [31:0]PC4_W_out;
wire [31:0]AO_W_out;
wire [31:0]DR_W_out;
wire [4:0]Forward_Addr_W_out;
wire [31:0]Forward_Data_W_out;

//W 级流水寄存器输出
wire stall;
//暂停信号
wire RegDst_D;
wire RegWrite_D;
wire jal_slt_D;
wire jal_slt_M;
//D 级控制信号

wire [1:0]sel_D_RS;
wire [1:0]sel_D_RT;
wire [1:0]sel_E_ALU1;
wire [1:0]sel_E_ALU2;
wire sel_M_WD;
//转发选择信号
wire [31:0]rs_final;
wire [31:0]rt_final;
wire [31:0]alu1_final;
wire [31:0]alu2_final;
wire [31:0]WD_final;
//MUX 输出信号
```

三、 测试程序

混合测试：

```
# 0x3000
lw $2 4($0)
# 0x3004
nop
# 0x3008
sw $4 12($0)
# 0x300c
lw $18 24($0)
# 0x3010
sw $20 28($0)
# 0x3014
nop
# 0x3018
sw $2 20($0)
# 0x301c
addu $19 $17 $6
# 0x3020
subu $11 $18 $12
# 0x3024
addu $3 $22 $21
# 0x3028
ori $30 $20 0x5050
# 0x302c
subu $9 $27 $13
# 0x3030
lw $21 20($0)
# 0x3034
addu $21 $23 $23
# 0x3038
lw $10 12($0)
# 0x303c
addu $6 $4 $17
# 0x3040
subu $5 $8 $14
# 0x3044
sw $16 20($0)
# 0x3048
```

```
addu $21 $8 $27
# 0x304c
nop
# 0x3050
subu $3 $22 $22
# 0x3054
lw $20 8($0)
# 0x3058
ori $23 $27 0x2424
# 0x305c
subu $4 $21 $8
# 0x3060
lw $26 8($0)
# 0x3064
nop
# 0x3068
```

ori

```
ori $t0,$zero,0xfffe
ori $t1,$zero,6
ori $t2,$t1,0xffff
ori $zero,$t1,5
```

lui & addu & subu

```
ori $t1,$0,5
ori $t2,$0,8
lui $t3,123
lui $t4,0xffff
ori $t4,$t4,0xffff
lui $t5,0xffff
ori $t5,$t5,0xfff8
```

```
addu $s0,$t2,$t1
addu $s1,$t4,$t2
```

```
addu $s2,$t4,$t5

subu $s3,$t2,$t1
subu $s4,$t1,$t2
subu $s5,$t2,$t4
subu $s6,$t4,$t5
```

beq

```
ori $t0,5
ori $t1,6
ori $t2,6

beq $t0,$t1,if1
ori $t3,7

if1:
addu $t4,$t1,$t3

beq $t1,$t2,if2
ori $t5,8

if2:
addu $t6,$t1,$t5
```

lw & sw

```
ori $t2,$0,8
lui $t3,123
lui $t4,0xffff
ori $t4,$t4,0xffff
lui $t5,0xffff
ori $t5,$t5,0xfff8

sw $t2,0($0)
sw $t3,4($0)
sw $t5,12($0)
```

```
lw $s0,8($0)
lw $s1,4($0)
lw $s2,0($0)
```

jr

```
ori $t0,$0,1
ori $t1,$0,5
ori $t2,$0,0
ori $t3,$0,1
addu $t0,$t0,$t0
addu $t2,$t2,$t3
beq $t2,$t1,end
ori $s1,$0,0x00003010
jr $s1
end:
```

jal

```
jal tag1
start:
jal end
tag1:
ori $t0,$0,0
jal tag2
ori $t1,$0,0
tag2:
ori $t2,$0,0
jal start
ori $t3,$0,0
end:
```

思考题

1. 在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来

A：

满足供给-需求模型的前后两条指令均会产生冒险冲突。

供给者有 CALR,CALI, LOAD, JAL

它们的会修改寄存器的值

需求者有 CALR,CALI ,BEQ, L, SAVE,

J 类指令既不是需求者也不是供给者

从这里设计暂停和转发的控制单元，首先 转发是为了在任何时刻检测到冲突的时候都能够将供给者要写入的数据发给需求者，而暂停是为了保证在任何时刻，转发的数据都是正确的。因此，当我们检测到转发无法解决时间冲突的问题的时候，我们就应当暂停，在暂停的时候，冻结 PC 与 D 级流水，清空 E 级流水，相当于插入了一条 n o p 指令。

```
ori $t1,6
beq $t0,$t1,if1
ori $t3,7
if1:
addu $t4,$t1,$t3
```

例如这里 如果不发生转发 则对于 addu，则无法读到正确的$t3 的值，所以程序就会发生错误。而有的时候对于后面指令需求的结果，若前面指令还未算出，则需要暂停一周期或多个周期。

转发总共有 6 种，E->D,M->D,W->D,M->E,W->E,W->M,则需要对应的设计六个转发的多选其对于数据进行选择，控制信号由转发控制器给出，如果不需要转发，则取原来的值，如果需要转发，则采取后面流水级回写的值。

由于 W 与 D 在空间上在同一个位置，所以实际上只需要设计 5 个转发器，W 到 D 的转发在 GRF 的内部实现即可。