

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
ins_r[31:0]	I	当前 E 级取数指令
ins_w[31:0]	I	当前 W 级回写指令
pc4_add[31:0]	I	输入W 级指令 pc+4的值
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
Regdata[31:0]	I	要回写入寄存器的数据
RegAddr[4:0]	I	要回写入寄存器的地址
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
ALUOp[4:0]	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
overflow	O	溢出标志位，1 代表溢出

B 功能描述 ALUOp[4:0]

功能名称	描述 Result=
0 00000 加	A+B
1 00001 减	A-B
2 00010 与	A&B
3 00011 或	A B
4 00100 非	~A
5 00101 逻辑左移	A<<sa
6 00110 逻辑右移	A>>sa
7 00111 算术右移	\$signed(A)>>>sa
8 01000 可变逻辑左移	B<<A[4:0]
9 01001 可变算术右移	B>>A[4:0]
10 01010 可变逻辑右移	\$signed(B)>>>A[4:0]
11 01011 逻辑与	A&B
12 01100 逻辑或	B A
13 01101 逻辑或非	~(B A)
14 01110 异或	B^A
15 01111 有符号小于置1	(\$signed(A)<\$signed(B)) ? 1:0
16 10000 无符号小于置1	(A<B) ? 1:0

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， $PC = PC + 4 + \text{sign_extend}(\text{immediate} \parallel 0^2)$
3	Jump	在时钟上升沿到达时，若jump信号有效，且reset信号无效， $PC = PC[31..28] \parallel 26\text{add} \parallel 0^2$
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]	0	机器码0-25位
Immediate[15:0]	0	机器码0-15位
Func[5:0]	0	机器码位0-5位
Sa[5:0]	0	机器码6-10位
Rd[5:0]	0	机器码11-15位
Rt[5:0]	0	机器码16-20位
Rs[5:0]	0	机器码21-25位
Ocode[5:0]	0	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
op[1:0]	I	选择存储指令类型
BE[3:0]	I	各个字节写使能
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

op	指令
0 00	sw
1 01	sh
2 10	sb

C 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. BE_EXT && MEM_EXT

信号名	方向	描述
A0[1:0]	I	输入低两位地址
op[1:0]	I	输入指令类型，op 码与 DM 相同
BE[4:0]	O	输出字节写使能

信号名	方向	描述
A0[1:0]	I	输入低两位地址
mem_op[2:0]	I	输入指令类型
Din[31:0]	I	输入 DM 读取数据
Dout[31:0]	O	输出最终写入寄存器的数据

mem_op	指令
0 000	lw
1 001	lbu
2 010	lb
3 011	lhu
4 100	lu

8. MULT_DIV

乘除运算模块

信号名	方向	描述
clk	I	时钟信号
D1[31:0]	I	输入指令类型
D2[31:0]	I	输入 DM 读取数据
op[3:0]	I	输出最终写入寄存器的数据
busy	O	输出繁忙暂停信号
HI[31:0]	O	hi 寄存器输出
LO[31:0]	O	lo 寄存器输出

mult_div_op	指令
0 0000	none
1 0001	mult
2 0010	div
3 0011	multu
4 0100	divu
5 0101	mfhi
6 0110	mflo
7 0111	mthi
8 1000	mtlo

9. 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	0	是否执行 j 指令
RegDst	0	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	0	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	0	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	0	寄存器堆写使能信号, 高电平有效
MemWrite	0	数据存储器写使能信号, 高电平有效
PC_sel	0	决定 PC 是否进行 branch 跳转, 高电平有效
EXT0[1:0]	0	决定 EXT 进行何种拓展
ALU0[2:0]	0	决定 ALU 进行何种运算
Sll_slt	0	选择 ALU 的第一个操作数, 若为 0, 为 rs, 若为 1, 为 rt
jr_slt	0	是否执行 jr 指令的选择信号
jal_slt	0	是否执行 jal 指令的选择信号
jalr_slt	0	是否执行 jalr 指令的选择信号
is_load	0	是否为取数型指令
is_save	0	是否为存储型指令
is_cal_r	0	是否为 R 类运算型指令
is_cal_i	0	是否为 I 类运算型指令
is_mu_di	0	是否为乘除型指令
is_branch_rs	0	是否为 b 类跳转使用 rs 型指令
is_branch_rsrt	0	是否为 b 类跳转使用 rs、rt 型指令
is_jalr	0	是否为 jalr 指令
is_mt	0	是否为写 hi lo 指令
is_mf	0	是否为读 hi lo 指令
is_jr	0	是否为读 jr 指令

B 各指令的控制信号

指令	Ju mp	RegD st	ALUS rc	Memt oReg	RegW rite	MemW rite	PC_s el	EX To	AL Uo	Sll_ slt	jr_s lt	jal_ slt
Ad du	0	1	0	0	1	0	0	00	00 0	0	0	0
Su bu	0	1	0	0	1	0	0	00	00 1	0	0	0
Sll	0	1	0	0	1	0	0	00	10 1	1	0	0
Or i	0	0	1	0	1	0	0	00	01 1	0	0	0
Lu i	0	0	1	0	1	0	0	10	00 0	0	0	0
Be q	0	0	0	0	0	0	1	00	00 1	0	0	0
Lw	0	0	1	1	1	0	0	01	00 0	0	0	0
Sw	0	0	1	0	0	1	0	01	00 0	0	0	0
J	1	0	0	0	0	0	0	00	00 0	0	0	0
jal	1	0	0	0	1	0	0	00	00 0	0	0	1
jr	0	0	0	0	0	0	0	00	00 0	0	1	0

指令分类:

load	save	cal_r	cal_i	mult/div	branch/rs	branch rs/rt	jlr	mt	mf
LB	SB	ADD	ADDI	MULT	BLEZ	BEQ	JALR	MTHI	MFHI
LBU	SH	ADDU	ADDIU	MULTU	BGTZ	BNE		MTLO	MFLO
LH	SW	SUB	ANDI	DIV	BLTZ				
LHU		SUBU	XORI	DIVU	BGEZ				
LW		SRL	ORI						
		SRA	SLTI						
		SLLV	SLTIU						
		SRLV	LUI						
		SLL							
		SRAV							
		AND							
		OR							
		NOR							
		XOR							
		SLT							
		SLTU							

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	mf 1/rd	cal_i 1/rt	load 2/rt	load 1/rt
branchl	rs	0					
jr	rs	0					
jalr	rs	0					
cal_r	rs/rt	1					
mu_div	rs/rt	1					
load	rs	1					
save	rs	1					
cal_i	rs	1					
mt	rs	1					
branch2	rs/rt	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	0	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_ALU2[1:0]	O	E alu2选择信号
sel_M_WD	O	M wd选择信号

10. CP0 协处理器

信号名	方向	描述
eret	I	W级回写数据
mtc0	I	M级回写数据
WRITE_data[31:0]	I	选择信号
Addr[4:0]	I	最终数据
clk	I	时钟信号
reset	I	复位信号
PC4[31:0]	I	E 级 PC+4
BD	I	M 级 BD 位
{T0_irq,T1_irq}	I	外部中断信号
EXC_code[4:0]	I	异常编码
NPC[31:0]	O	CP0 输出跳转 CP 值
READ_data[31:0]	O	CP0 寄存器的读取
CP0_jump	O	CP0 跳转是否执行标志位

当检测到异常或中断的时候，跳转至 0x00004180
处理结束后跳回

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;
```

三、 测试程序

完整测试程序见附件

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
```

完整测试见附件

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 $t5,$t5,0xffff8
```

```
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 $t5,$t5,0xffff8
```

```
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. 我们计组课程一本参考书目标题中有“硬件/软件接口”接口字样，那么到底什么是“硬件/软件接口”？

A: 我认为是一个硬件、软件交接结合的部分，在这一部分中，硬件需要完成一部分的工作，软件需要完成一部分的工作，两者要共同设立一些规范，就好比 CP0 里面的寄存器，有的位是硬件写入，有的位是软件写入，硬件与软件共同协作，完成从硬件到软件的过渡。

2. 在我们设计的流水线中，DM 处于 CPU 内部，请你考虑现代计算机中它的位置应该在何处。

A: 我认为应该像 timer 一样，处在 CPU 外部的位。

3. BE 部件对所有的外设都是必要的吗？

A: 我认为不是，例如 timer 中的寄存器，不能使用 sh、sb 去写入，也不能使用 lb、lh 读出，因此不需要。

4. 请开发一个主程序以及定时器的 exception handler。整个系统完成如下功能

A: <1>.保存现场：在异常处理例程入口，需要保护被中断的程序的现场，存储寄存器的状态，保证关键状态不被覆盖。一般用 k0 和 k1 这两个寄存器索引一块内存区域，用来存储其他的寄存器。这块内存区域一般被称作中断栈（Interrupt Stack），用于存储寄存器状态，并且支持复杂的 C 等高级语言编写的异常处理例程。

<2>.处理异常：根据 C0_CAUSE:ExcCode 确定发生了什么类型的异常，然后调用 OS 定义的不同的函数处理。假设此时再其他异常的入口点（excNormalVec=0x80000180），若确定了是中断（C0_CAUSE:ExcCode=0），则进一步调用中断处理函数 excIntStub；否则调用 excStub。

<3>.准备返回：恢复现场，修改 C0_SR，设置成安全模式（内核态，禁止异常），也就是异常发生后的模式。

<4>.从异常返回：控制权交给异常 victim 指令，将内核特权级恢复为较低的特权级。MIPS CPU 提供了原子指令“eret”做这个工作：既清除 C0_SR:EXL 位，也将控制权返回给存储在 EPC 中的地址。

见附件

5. 请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？

A: 是以中断的形式被 CPU 知道并且处理的