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.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

二、 数据通路设计

需要定义的线网型变量如下：

```
wire [5:0] op;
wire [5:0] func;
wire [25:0] j_addr;
wire [15:0] imm;
wire [4:0] rs;
wire [4:0] rt;
wire [4:0] rd;
wire [4:0] sa;
//IM 中指令各个字段

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;
//CONTROLLER 输出的控制信号

wire [31:0] alu_res;
wire Ztag;
//ALU 输出的运算结果

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
```

用其在子模块实例化中将各个子模块连接，例如：

```
CONTROL control(
        .op(op),
        .func(func),
        .jump(jump),
        .RegDst(RegDst),
        .ALUSrc(ALUSrc),
        .MemtoReg(MemtoReg),
        .RegWrite(RegWrite),
        .MemWrite(MemWrite),
        .branch(branch),
        .extop(extop),
        .aluop(aluop),
        .sll_slt(sll_slt),
        .jr_slt(jr_slt),
        .jal_slt(jal_slt)
    );



ALU alu(
       .rs_out(grfo_rs),
       .rt_out(grfo_rt),
       .imm_ext(imm_ext),
       .sa(sa),
       .aluop(aluop),
       .sll_slt(sll_slt),
       .ALUSrc(ALUSrc),
       .result(alu_res),
       .zero(Ztag)
    );
```

三、 测试程序

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

L0.T2

1. 根据你的理解，在下面给出的 DM 的输入示例中，地址信号 addr 位数为什么是[11:2]而不是[9:0]？这个 addr 信号又是从哪里来的？

   因为 MIPS 是按字节编址，所以按字寻址的时候低两位是为了确定要取得是一个字中四个字节的哪一个，又因为有 1024 个所以为 11：2。

   addr 来自 alu 的运算结果输出，因为对于 dm 来说只有 sw 指令会用到，要写的数据通过取出后在 alu 中与 0 进行加法运算后输出，写入 DM 中。

2. 在相应的部件中，**reset 的优先级**比其他控制信号（不包括 clk 信号）都要**高**，且相应的设计都是**同步复位**。清零信号 reset 是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

   寄存器与内存，包括 GRF 的 32 个寄存器，PC，DM。

   因为这些部件可以记录电路之前的状态并且如果不手动进行清零就会一直保持这种状态，所以要清零电路的时候，要对这些部件进行清零处理。

L0.T4

1. 列举出用 Verilog 语言设计控制器的几种编码方式（至少三种），并给出代码示例。

①对于每个信号进行编码：

例子：

```
assign RegDst=((op==6'b000000&&func==6'b100001)||
 (op==6'b000000&&func==6'b100011)||(op==6'b000000&&func==6'b000000))?1:0;
```

②对于每个指令进行编码：

例子：

```
if(addu==1)begin
    jump=0;
    RegDat=1;
    ALUSrc=0;
    MemtoReg=0;
    RegWrite=1;
    MemWrite=0;
    ……
end
```

③case（op）
000000：……

2. 根据你所列举的编码方式，说明他们的优缺点。
①代码简洁，易于添加，但是如果不加注释的话对于各个指令的识别较差。
②容易识别，但是代码量十分大
③既不容易识别，代码量又大……

L0.T5

1．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。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分 。

addi 与 addu、add 与 addu 的区别：当 add 运算结果算术溢出时，会触发算术溢出异常并触发中断，addu 则会继续运行。

3. 根据自己的设计说明单周期处理器的优缺点。
单周期处理器优点：设计简单
缺点：由于时钟周期由最长的一条指令的运行时间决定，因此会造成时间的浪费

4.简要说明 jal、jr 和堆栈的关系。

类比于 C 程序，jal 相当于压栈，jr 相当于出栈，jal 指令就如同调用递归函数/子过程，jr 相当于“return”。