Verilog流水线CPU设计报告(P5)

一，CPU设计方案综述

（一）总体设计概述

本设计为由Verilog实现的最多支持1024条指令的五级流水线MIPS-CPU，支持addu，subu，ori，lw，sw，beq，lui，nop, jr, jal和j 这十一条指令。为了实现这这些功能，本实验设计出的CPU被分为了IFU（指令存储单元），IF_ID（if/id级流水线寄存器1），GRF（寄存器组），ID_EX（id/ex级流水线寄存器），ALU（算术逻辑运算单元），EX_MEM（ex/mem级流水线寄存器），DM（数据存储器），MEM_WB（mem/wb级流水线寄存器），controller （控制单元），Stall（暂停控制），Trans（转发控制）这11个模块。

（二）关键模块定义

1.GRF

A 介绍

通用寄存器组，用于临时存放和调用数据，数据为32位。

B 端口定义

寄存器堆

端口	输入输出(位宽)	描述
RA1	I [4:0]	指定寄存器堆中的一个，将其中数据输出到RD1
RA2	I [4:0]	指定寄存器堆中的一个，将其中数据输出到RD2
WA	I [4:0]	指定32个寄存器中的一个，写入WD数据
WD	I [31:0]	输入的数据
reset	I	异步复位端口
clk	I	时钟信号
WE	I	WE为1：可写入；WE为0，不可写入
RD1	O [31:0]	RA1指定寄存器的值
RD2	O [31:0]	RA2指定寄存器的值
PC	O [31:0]	当前PC值

C 功能介绍

功能名称	功能描述
读入数据	当WE为1时，可以将WD写入WA指定的寄存器中
输出数据	可以将RA1和RA2指定的寄存器中的值通过RD1和RA2输出
复位	当reset信号为1时，实现异步复位

2.DM

A 介绍

数据存储器，可以根据指令存放和调用数据。

B 端口定义

端口	输入输出（位宽）	描述
addr	I [5:0]	待写入（输出）地址
PC	I [31:0]	当前PC值
din	I [31:0]	待写入数据
reset	I	异步复位信号
clk	I	时钟信号
Store	I	写入使能端
IR	I [31：0]	当前指令
dout	O [31:0]	输出数据

C 功能定义

功能名称	功能描述
读入数据	当WE为1且时钟上升沿到来时，写入数据
输出数据	将WA指定的内存中存储的数据输出
复位	当reset信号为1时，实现异步复位

3.ALU

A 介绍

算数逻辑运算单元，根据提供的数据实现指令执行过程中所需要的数学运算。

B 端口定义

端口	输入输出（位宽）	描述
RD1	I [31:0]	grf提取的寄存器的第一个值
RD2	I [31:0]	grf提取的寄存器的第二个值
imm	I [2:0]	id_ex传递的对立即数处理之后的版本
PC	I [31:0]	当前PC值
IR	I [31:0]	当前指令
C	O [31:0]	运算结果
zero	O	C是0为1，否则为0

C 功能定义

功能名称	功能描述
数学运算	根据提供的数据实现指令执行过程中所需要的数学运算
判0	判断运算结果是否为0

4.IFU

A 介绍

指令存储单元，内部涵盖指令计数器（PC）和指令寄存器（IM），用于取出当前执行的指令和存放下一个指令。

B 端口定义

端口	输出输出（位宽）	描述
nextPC	I [31:0]	下一条指令地址
clk	I	时钟信号
reset	I	异步复位信号
En	I	暂停控制信号
intr	O [31:0]	从IM中取出的当前指令
PC	O [31:0]	当前指令地址

C 功能定义

功能名称	功能描述
取指令	将当前指令取出用于后续逻辑处理
存指令	将处理后的写一条指令地址存入nextPC
复位	当reset信号为1，PC回归32b'0
暂停	当暂停信号有效的时候，暂停对PC操作

5.control

A 介绍

根据指令的操作码和功能码识别指令。

B 端口定义

端口	输出输出（位宽）	描述
op	I [5:0]	操作码
fuc	I [5:0]	功能码
addu	O	addu信号控制
subu	O	subu信号控制
lw	O	lw信号控制
sw	O	sw信号控制
beq	O	beq信号控制
lui	O	lui信号控制
jal	O	jal信号控制
jr	O	jr信号控制
j	O	j信号控制
ori	O	ori信号控制

C 功能定义

功能名称	功能描述
生成控制	根据指令的机器码分辨指令

6.IF_ID

A 介绍

第一级流水线寄存器，用于传输PC和指令机器码。

B 端口定义

端口	输出输出（位宽）	描述
clk	I	时钟信号
reset	I	重置信号
En	I	暂停控制
intr	I [31:0]	指令
PC	I [31:0]	PC值
PCctrl	I [1:0]	PC控制信号
if_id_PC	O reg [31:0]	PC寄存器
IR	O reg [31:0]	指令寄存器
if_id_PCctrl	O reg [1:0]	PC控制信号寄存器

C 功能定义

功能名称	功能描述
寄存数据	将if级处理出来的数据和控制信号存在寄存器中

7.ID_EX

A 介绍

第二级流水线寄存器，主要用于传输用于指令运算的寄存器值。

B 端口定义

端口	输出输出（位宽）	描述
clk	I	时钟信号
reset	I	重置信号
clear	I	暂停时的清空信号
IR	I [31:0]	指令
PC	I [31:0]	PC的值
RD1	I [31:0]	grf提取的第一个寄存器值
RD2	I [31:0]	grf提取的第二个寄存器值
id_ex_RD1	O reg [31:0]	存RD1寄存器
id_ex_RD2	O reg [31:0]	存RD2寄存器
imm	O reg [31:0]	存立即数扩展后结果
id_ex_WA	O reg [4:0]	存grf修改地址寄存器
id_ex_PC	O reg [31:0]	PC值寄存器
id_ex_IR	O reg [31:0]	指令寄存器

C 功能定义

功能名称	功能描述
寄存数据	将if级处理出来的数据和控制信号存在寄存器中
译码	将指令译码，得到立即数和grf存储地址的处理结果

8.EX_MEM

A 介绍

第三级流水线寄存器，主要用于传递ALU运算得到的结果。

B 端口定义

端口	输出输出（位宽）	描述
clk	I	时钟信号
reset	I	重置信号
ALUout	I [31:0]	ALU单元运算得到的值
IR	I [31:0]	指令
PC	I [31:0]	PC的值
RD2	I [31:0]	grf提取的第二个寄存器值
WA	I [4:0]	寄存器存储的地址
ex_mem_ALUout	O reg [31:0]	存ALUout寄存器
ex_mem_din	O reg [31:0]	存写入MD的数值的寄存器
ex_mem_WA	O reg [31:0]	存grf修改地址的寄存器
ex_mem_IR	O reg [4:0]	存传递的指令的寄存器
ex_mem_PC	O reg [31:0]	PC值寄存器

C 功能定义

功能名称	功能描述
寄存数据	将if级处理出来的数据和控制信号存在寄存器中

9.MEM_wb

A 介绍

第四级流水线寄存器，主要用于传输MD输出的值以及产生grf写入的控制信号。

B 端口定义

端口	输出输出（位宽）	描述
clk	I	时钟信号
reset	I	重置信号
ALUout	I [31:0]	ALU单元运算得到的值
IR	I [31:0]	指令
PC	I [31:0]	PC的值
dout	I [31:0]	MD题取的值
WA	I [4:0]	寄存器存储的地址
ex_mem_WD	O reg [31:0]	存写入grf数值的寄存器
ex_mem_RegWrite	O reg	存写入grf控制信号的寄存器
ex_mem_WA	O reg [4:0]	存grf修改地址的寄存器
ex_mem_IR	O reg [31:0]	存传递的指令的寄存器
ex_mem_PC	O reg [31:0]	PC值寄存器

C 功能定义

功能名称	功能描述
寄存数据	将if级处理出来的数据和控制信号存在寄存器中
译码	将指令译码，得到存入grf的数值和存储地址的处理结果

10.Stall（主）

A 介绍

暂停控制信号生成器。

B 端口定义

端口	输出输出（位宽）	描述
id_IR	I [31:0]	id级的指令
ex_IR	I [31:0]	ex级的指令
mem_IR	I [31:0]	mem级的指令
stop	O	输出暂停信号

C 功能定义

功能名称	功能描述
产生控制信号	产生暂停的控制信号，在主模块中生成暂停信号

Stall--get_tuse

A 介绍

产生id级指令的get_tuse值。

B 端口定义

端口	输出输出（位宽）	描述
IR	I [31:0]	当前指令
rs_tuse	O [1:0]	rs寄存器的tuse
rt_tuse	O [1:0]	rt寄存器的tuse

C 功能定义

功能名称	功能描述
产生tuse值	产生id级指令的tuse值，用于判断是否要暂停

Stall--get_tnew

A 介绍

产生ex和mem级指令的get_tnew值。

B 端口定义

端口	输出输出（位宽）	描述
IR	I [31:0]	当前指令
is_ex	I	是否是ex级
cg_reg	O [4:0]	当前级修改的寄存器
tnew	O [1:0]	获得的tnew值

C 功能定义

功能名称	功能描述
产生tnew值	产生ex和mem级指令的tnew值，用于判断是否要暂停
产生被修改的寄存器编号	产生ex和mem级当前指令修改的寄存器编号

11.Trans

A 介绍

用于产生转发的控制信号。

B 端口定义

端口	输出输出（位宽）	描述
id_IR	I [31:0]	id级的指令
ex_IR	I [31:0]	ex级的指令
mem_IR	I [31:0]	mem级的指令
wb_IR	I [31:0]	wb级的指令
ex_WA	I [4:0]	ex级修改寄存器的编号
mem_WA	I [4:0]	mem级修改寄存器的编号
wb_WA	I [4:0]	wb级修改的寄存器的编号
F_id_rs	O [1:0]	id级rs位置grf值转发信号
F_id_rt	O [1:0]	id级rt位置grf值转发信号
F_ex_rs	O [1:0]	ex级rs位置grf值转发信号
F_ex_rt	O [1:0]	ex级rt位置grf值转发信号
F_mem_rt	O [1:0]	mem级rt位置grf转发信号

C 功能定义

功能名称	功能描述
产生转发信号	产生控制转发的信号

（三）重要机制及其实现方法

1 暂停

可以使用计算tnew和tuse的方法。

tnew定义：一个修改grf的指令在从当前级开始计算，最少还有多少个时钟周期才会稳定产生写入寄存器的值。

tuse定义：一个使用grf中值的指令在从当前级开始计算，最多还有多少个时钟周期才会要使用到目标寄存器中的值。

一般将所有指令在id级时就判断是否需要暂停，即计算出id级指令的tuse，ex和mem级的tnew，以及id级指令使用的寄存器rs，rt和ex和mem级指令写入寄存器cg_reg。这样容易得到暂停判断的逻辑（核心是tnew>tuse)。

```
assign stop = (id_IR[`rs]==ex_cg_reg && id_IR[`rs]!=0 && rs_tuse<ex_tnew )||(id_IR[`rs]==mem_cg_reg && id_IR[`rs]!=0 && rs_tuse<mem_tnew)||
              (id_IR[`rt]==ex_cg_reg && id_IR[`rt]!=0 && rt_tuse<ex_tnew )||(id_IR[`rt]==mem_cg_reg && id_IR[`rt]!=0 && rt_tuse<mem_tnew );
```

注意需要特判一下id级即将使用的寄存器是否为0号寄存器，如果是，则不需要转发，因为0号寄存器无论怎么操作都不会改变它的值。

二，测试方案

使用c++程序编程实现测试数据的生成。

```
#include <bits/stdc++.h>

using namespace std;

vector<int> r;
mt19937 mt(time(0));
uniform_int_distribution<int>
    imm16(0, (1 << 16) - 1),
    siz(0, 1023),
    reg(0, 2),
    grf(1, 30),
    I(1, 6),
    J(7, 9),
    IJ(1, 9);

int cnt;

void solve(int);

int getR(){
    return r[reg(mt)];
}

void addu(){
    printf("addu $%d, $%d, $%d\n", getR(), getR(), getR());
}

void subu(){
    printf("subu $%d, $%d, $%d\n", getR(), getR(), getR());
}

void ori(){
    printf("ori $%d, $%d, %d\n", getR(), getR(), imm16(mt));
}

void lw(){
    printf("lw $%d, %d($0)\n", getR(), siz(mt) * 4);
}

void sw(){
    printf("sw $%d, %d($0)\n", getR(), siz(mt) * 4);
}

void lui(){
    printf("lui $%d, %d\n", getR(), imm16(mt));
}

void beq(){
    printf("beq $%d, $%d, label%d\n", getR(), getR(), ++cnt);
    solve(I(mt));
    solve(I(mt));
    solve(I(mt));
    printf("label%d: ", cnt);
    solve(I(mt));
}
```

```
void jaljr(){
    int x = getR();
    printf("ori $%d $0 16\n",x);
    printf("jal label%d\n", ++cnt);
    solve(I(mt));
    solve(I(mt));
    printf("label%d:", cnt);
    printf("addu $%d, $%d, $31\n", x, x);
    printf("jr $%d\n", x);
    solve(I(mt));
}

void j(){
    printf("j label%d\n", ++cnt);
    solve(I(mt));
    solve(I(mt));
    solve(I(mt));
    printf("label%d: ", cnt);
    solve(I(mt));
}

void solve(int i){
    switch(i){
        case 1:
            addu();
            break;
        case 2:
            subu();
            break;
        case 3:
            ori();
            break;
        case 4:
            lw();
            break;
        case 5:
            sw();
            break;
        case 6:
            lui();
            break;
        case 7:
            beq();
            break;
        case 8:
            jaljr();
            break;
        case 9:
            j();
            break;
    }
}

int main(){
    r.push_back(grf(mt)), r.push_back(grf(mt)), r.push_back(grf(mt));
    freopen("test.asm", "w", stdout);
    puts("ori $28, $0, 0");
```

```
    puts("ori $29, $0, 0");
    for(int i = 1;i <= 700;i++){
        int x = IJ(mt);
        if(x > 6) i += 5;
        solve(x);
    }
}
```

3 自动化测试方案

先用上述程序生成测试代码写入test.asm，将test.asm导入魔改mars，输出到m.out，再将机器码导出到code.txt，再利用iverilog将test.v的输出写入v.out,最后分别用sort.exe和sortm.exe对文件进行排序，比对，将比对结果输出到check.txt，跳回最开始开始循环。

```
:goon
gen.exe
java -jar Mars_Changed.jar db mc CompactDataAtZero nc test.asm > m.out
java -jar Mars_Changed.jar mc CompactDataAtZero a dump .text HexText code.txt test.asm
iverilog -o tb.out -y D:\verilog\P5-2 D:\verilog\P5-2\test.v
vvp tb.out -v > v.out
sort.exe
m_sort.exe
fc aftersort.out maftersort.out >> check.txt
goto goon
```

三，思考题

（一）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?

文件	模块接口定义

dm.v	dm(clk,reset,MemWrite,addr,din,dout);
	input clk; //clock
	input reset; //reset
	input MemWrite; //memory write enable
	input [11:2] addr; //memory's address for write
	input [31:0] din; //write data
	output [31:0] dout; //read data

- 因为DM中存储器一个字占一个地址，然而mips-cpu中默认一个字节占一个地址，所以这里需要把DM输入地址（ALUout）的11到2位作为输出，相当于对地址做一个除4操作。
- addr信号的来源固定是算术逻辑运算单元的输出ALUout。

（二）思考Verilog语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣。

- 一共有两种方式
    - **指令对应的控制信号如何取值**:

```
if(ori) begin
    WDctrl=0;
    ALUctrl=2;
    MemWrite=0;
    RegWrite=1;
    RegDst=0;
    ALUOp=2;
    PCctrl=3;
end
```

这种方式比较有针对性，对每个指令数据通路需求的控制信号展现的较为全面，不容易出错，但是修改起来比较麻烦，不够简洁。再后续添加指令后容易使代码变得冗长。

- **控制信号每种取值所对应的指令**:

```
assign ALUctrl=(ori)? 2:
               (lw||sw)?1:
                0;
```

这种方式比较简便，利用了真值表的方法避免了穷举的麻烦，而且后续经行修改添加的操作时也十分简便，但是这种方法比较容易出错。

（三）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所驱动的部件具有什么共同特点?

- 这些部件都具有时序特征，可以对数据进行存储和提取，都可以在时钟上升沿到来时，在对应的位置（地址）存储输入的值。

（四）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 。

- 因为addi相对于addiu，add相对于addu多判断了一个结果是否溢出，所以在忽略溢出的情况下，add和addu，addi和addiu是等价的。

（五）根据自己的设计说明单周期处理器的优缺点

- 优点：逻辑相对比较简单，易于实现，处理结果比较精确，不容易出错。
- 缺点：效率比较低，运行指令速度较慢，无法适应工程化要求。