Verilog流水线CPU设计报告

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。

```
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 );
```

主要的判断逻辑是$tuse > tnew$。

2 转发

可使用枚举分析的方法。

易知需要利用寄存器值的地方在流水线中有五处，而可以提供数值的地方有三处。这样就可以列表分类枚举，判断能否由A处转发向B处的条件为，B处需要的寄存器与A处修改的寄存器相同，且A处tnew为0，且被改的寄存器不为0号寄存器。

具体分析方法用列表解决。

其中wb部分向id部分的转发可以通过内部转发实现。

转发				ID/EX	EX/MEM			MEM/wb			
				jal	R	I/lui	jal	jal	I/lui	load	R/I
流水级	源寄存器	指令	控制信号								
ID	rs	beq/jr	F_id_rs	PC+8	ALUout	ALUout	ALUout	内部转发			
	rt	beq	F_id_rt								
EX	rs	R/I/lui/lw/	F_ex_rs					wb_WD	wb_WD	wb_WD	wb_WD
	rt	R/sw	F_ex_rt								
MEM	rt	sw	F_mem_rt								

二，测试方案

（一）典型样例测试

```
jal label2
ori $1, $0, 12
addu $1, $7, $1
label2: addu $1, $1, $31
jr $1
sw $1, 3324($0)
ori $7, $7, 51698
lui $7, 16108
sw $1, 1648($0)
lw $7, 2480($0)
lw $7, 2240($0)
subu $10, $7, $1
ori $10, $1, 18076
jal label3
ori $1, $0, 12
addu $7, $10, $10
label3: addu $1, $1, $31
jr $1
lui $1, 26585
j label4
sw $1, 112($0)
sw $10, 532($0)
label4: subu $7, $1, $7
j label5
lw $1, 4072($0)
lw $1, 3648($0)
label5: lui $10, 38753
beq $7, $10, label6
addu $1, $10, $10
ori $10, $1, 16114
label6: ori $7, $10, 43960
ori $1, $10, 32805
lw $7, 1148($0)
jal label7
ori $1, $0, 12
subu $7, $7, $10
label7: addu $1, $1, $31
```

```
jr $1
ori $10, $7, 19780
subu $10, $7, $7
jal label8
ori $1, $0, 12
lui $7, 39971
label8: addu $1, $1, $31
jr $1
lw $1, 1588($0)
lui $10, 32775
subu $7, $1, $7
beq $1, $10, label9
sw $7, 972($0)
ori $10, $1, 8719
label9: lw $1, 4072($0)
beq $1, $1, label10
subu $7, $7, $10
subu $10, $1, $1
label10: sw $7, 544($0)
addu $7, $7, $7
ori $10, $7, 17477
beq $10, $7, label11
lw $1, 624($0)
lui $7, 12684
label11: addu $10, $1, $1
sw $7, 884($0)
jal label12
ori $7, $0, 12
sw $7, 2900($0)
label12: addu $7, $7, $31
jr $7
ori $7, $1, 49409
beq $7, $10, label13
subu $1, $1, $7
addu $10, $1, $7
label13: subu $1, $1, $10
ori $10, $10, 64944
sw $7, 1344($0)
j label14
```

```
j label14
ori $10, $10, 39892
addu $7, $1, $1
label14: subu $1, $1, $10
subu $10, $1, $7
sw $10, 220($0)
ori $7, $1, 22591
j label15
sw $1, 3856($0)
addu $1, $7, $1
label15: ori $7, $1, 65168
lui $10, 62746
subu $7, $1, $7
lui $1, 20237
lui $7, 3383
j label16
subu $1, $7, $10
lui $7, 4679
label16: ori $10, $1, 43712
```

为最大化冲突，只采用3种寄存器。

（二）自动化生成测试数据

c++编程生成测试数据：

```cpp
#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）在采用本节所述的控制冒险处理方式下，PC的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

- 数据通路：
  - PC的下一个值在id级就被计算出来，如果是直接跳转的指令，则直接根据公式计算下一个PC的值，如果是条件跳转指令和jr指令，则需要考虑id级访问到的寄存器的值，这里涉及到旁路转发，在id级设置比较器，使用转发后的寄存器值对下一个地址进行计算，并再下一个时钟上升沿更新PC值。如果是非跳转指令，则下一个PC直接等于$PC+4$。
- 控制信号：
  - PC有关的控制信号有PCctrl（选择跳转类型），F_id_rs（id级rs寄存器转发控制信号），F_id_rt（id级rt寄存器转发控制信号）后两者用于判断转发到最终使用的寄存器值的来源。

（二）对于jal等需要将指令地址写入寄存器的指令，为什么需要回写PC+8?

- 因为对于跳转指令要考虑延迟槽，PC+4的指令是默认在跳转前执行的，所以跳转后应该到PC+4的下一条也就是PC+8处。

（三）为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由ALU或者DM等部件来提供数据?

- 因为这样可以保证输出信号的稳定，降低流水线上的信号缓冲延迟，从而降低流水线的周期。

（四）“转发（旁路）机制的构造”中的Thinking 1-4；

Thinking 1：如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题.

- 会出现数据冲突，前一个修改寄存器的指令还未将值写入grf，后一个就将数据提取出来，这样便会导致提取了修改之前的数据，造成错误。如：

```
lw $t5 4($0)
addu $t6 $t5 $t5
```

Thinking 2：我们为什么要对GPR采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢?

- 如果对GPR采用内部转发机制会简化转发的过程，减少接线的浪费。如果不采用内部转发，也可以从写入数据端接线至输出端，按照一般的转发机制（包括控制信号的生成）来处理。

Thinking 3：为什么0号寄存器需要特殊处理?

- 因为0号寄存器永远不会改变它的值，永远是0，所以不能将数据转发至与使用0号寄存器值相关的指令，不然会导致错误。

Thinking 4：什么是“最新产生的数据”?

- 就是距离当前处理的指令最近且在它之前的指令产生的数据，也就是在流水线模型当中最靠左的部分产生的数据优先级最高。

（五）在AT方法讨论转发条件的时候，只提到了“供给者需求者的A相同，且不为0”，但在CPU写入GRF的时候，是有一个we信号来控制是否要写入的。为何在AT方法中不需要特判we呢？为了用且仅用A和T完成转发，在翻译出A的时候，要结合we做什么操作呢?

- AT法中，在id级就判断好此指令是否是修改寄存器的指令，所以不需要特判WE。
- 判断其是否是修改寄存器的指令，也就是WE是否是1，是则将A原值传递，否则将A置0（因为0号寄存器被特判，后续不会经行任何操作）。