Verilog流水线CPU设计报告(P6)

一，CPU设计方案综

（一）总体设计概述

本设计为由Verilog实现的最多支持4096条指令的五级流水线MIPS-CPU，支持MIPS-C3指令集50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），NPC（计算下一个地址），multi_div（乘除模块）这13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
具体指令译码(略50个)	O	指令译码匹配结果

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

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
ex_cg_reg	I [4:0]	ex指令修改的寄存器
mem_cg_reg	I [4:0]	mem指令修改的寄存器
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
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

12 NPC

A 介绍

用于产生下一个PC地址

B 端口定义

端口	输出输出（位宽）	描述
RD1	I [31:0]	grf读出的第一个值
RD2	I [31:0]	grf读出的第二个值
PC	I [31:0]	当前ifu的PC值
id_PC	I [31:0]	id级的PC值
IR	I [31:0]	ex级修改寄存器的编号
nPC	I [4:0]	mem级修改寄存器的编号

C 功能定义

功能名称	功能描述
产生下一个PC地址	产生下一个PC地址

13 mult_div

A 介绍

用于处理乘除法。

B 端口定义

端口	输出输出（位宽）	描述
clk	I	时钟信号
reset	I	复位信号
IR	I [31:0]	具体的指令
A	I [31:0]	参与运算的第一个值
B	I [31:0]	参与运算的第二个值
hi	O reg [31:0]	高位
lo	O reg [31:0]	低位
busy	O	busy信号，为说明需要暂停

C 功能定义

功能名称	功能描述
乘除	完成乘除操作
写读hi/lo	完成将数据写入和将数据读出hi/lo寄存器的操作

（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	l/lui	jal	jal	l/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								

3 乘除

主要使用乘除法模块来完成乘除操作，将hi，lo直接作为寄存器输出。

具体逻辑：如果检测到ex指令为乘除指令，将start置1，在下一个时钟上升沿到来时直接将结果计算出来并存入hi，lo寄存器，乘法将计数器cnt置5，除法将计数器置10，之后，没经过一个时钟上升沿技术器减1，直到为0。判断busy的条件为 (start&&cnt)，同时，外部模块检测到busy信号则执行暂停操作（此暂停完全等同于之前的暂停操作）。

二，测试方案

二，测试方案

（一）典型样例测试

```
lw $22, 8($4)
ori $24, $0, 0
lb $4, 13($24)
bltz $4, label1
mfhi $24
srl $22, $24, 17
label1: slti $22, $4, -22479
j label2
mult $22, $4
multu $24, $22
label2: and $24, $24, $4
sltu $24, $22, $22
sllv $24, $4, $24
mult $4, $4
multu $22, $22
sltu $24, $22, $24
subu $24, $4, $24
ori $24, $24, 1
div $4, $24
ori $22, $0, 0
lbu $22, 14($22)
bltz $24, label3
subu $24, $24, $24
sub $24, $4, $4
label3: sub $4, $24, $24
addiu $22, $22, 5387
mthi $24
ori $22, $22, 1
divu $4, $22
srlv $4, $4, $4
xori $24, $4, 55792
ori $22, $4, 40029
sra $4, $24, 7
addi $22, $24, 0
ori $4, $0, 0
lhu $4, 4($4)
bgez $22, label4
add $4, $0, $4
xori $24, $24, 45460
```

```
label4: ori $4, $0, 0
sh $4, 2($4)
and $22, $4, $22
bne $24, $22, label5
ori $24, $0, 0
lb $24, 1($24)
ori $4, $0, 0
lb $24, 15($4)
label5: ori $22, $0, 0
sw $4, 12($22)
bltz $4, label6
sltiu $4, $24, -270
slt $4, $24, $4
label6: ori $4, $4, 1
divu $4, $4
mfhi $22
srl $4, $22, 15
blez $22, label7
ori $22, $0, 0
lbu $22, 7($22)
multu $4, $4
label7: ori $4, $0, 0
sh $4, 14($4)
bne $24, $24, label8
lui $22, 64326
ori $24, $0, 0
lh $22, 4($24)
label8: xor $24, $22, $4
j label9
lui $22, 8375
srav $22, $24, $22
label9: ori $22, $0, 0
sb $24, 11($22)
bltz $22, label10
lui $4, 53387
sltiu $4, $24, 6033
label10: sub $24, $4, $4
beq $22, $22, label11
ori $4, $0, 0
```

```
ori $4, $0, 0
lh $4, 4($4)
ori $22, $0, 0
lhu $24, 8($22)
label11: lui $4, 46646
bltz $4, label12
sllv $24, $22, $22
srlv $22, $24, $22
label12: and $4, $22, $22
ori $4, $0, 0
lb $22, 8($4)
sltiu $24, $22, 18276
ori $4, $0, 0
lb $22, 6($4)
ori $24, $22, 58579
ori $4, $4, 1
divu $4, $4
subu $4, $22, $24
bne $4, $4, label13
xori $4, $22, 27607
sltu $22, $24, $4
label13: sub $22, $4, $4
```

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>
    u16(0, (1 << 16) - 1),
    s16(-(1 << 15), (1 << 15) - 1),
    siz(0, 15),
    reg(0, 2),
    grf(1, 30),
    shift(0, 31),
    I(1, 40),
    J(41, 49),
    IJ(1, 49),
```

```
    one(11, 40);

int cnt, tot;

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

void solve(int i){
    int x, X;
    switch(i){
        case 1:
            x = getR();
            printf("ori $%d, $0, 0\n", x);
            printf("lb $%d, %d($%d)\n", getR(), siz(mt), x);
            tot += 2;
            break;
        case 2:
            x = getR();
            printf("ori $%d, $0, 0\n", x);
            printf("lbu $%d, %d($%d)\n", getR(), siz(mt), x);
            tot += 2;
            break;
        case 3:
            x = getR();
            printf("ori $%d, $0, 0\n", x);
            printf("lh $%d, %d($%d)\n", getR(), siz(mt) >> 1 << 1, x);
            tot += 2;
            break;
        case 4:
            x = getR();
            printf("ori $%d, $0, 0\n", x);
            printf("lhu $%d, %d($%d)\n", getR(), siz(mt) >> 1 << 1, x);
            tot += 2;
            break;
        case 5:
            x = getR();
            printf("ori $%d, $0, 0\n", x);
            printf("lw $%d, %d($%d)\n", getR(), siz(mt) >> 2 << 2, x);
            tot += 2;
            break;
        case 6:
            x = getR();
            printf("ori $%d, $0, 0\n", x);
            printf("sb $%d, %d($%d)\n", getR(), siz(mt), x);
            tot += 2;
            break;
        case 7:
            x = getR();
            printf("ori $%d, $0, 0\n", x);
            printf("sh $%d, %d($%d)\n", getR(), siz(mt) >> 1 << 1, x);
            tot += 2;
            break;
        case 8:
            x = getR();
            printf("ori $%d, $0, 0\n", x);
            printf("sw $%d, %d($%d)\n", getR(), siz(mt) >> 2 << 2, x);
            tot += 2;
```

```
            break;
        case 9:
            x = getR();
            printf("ori $%d, $%d, 1\n", x, x);
            printf("div $%d, $%d\n", getR(), x);
            tot += 2;
            break;
        case 10:
            x = getR();
            printf("ori $%d, $%d, 1\n", x, x);
            printf("divu $%d, $%d\n", getR(), x);
            tot += 2;
            break;
        case 11:
            printf("add $%d, $0, $%d\n", getR(), getR());
            tot++;
            break;
        case 12:
            printf("addu $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 13:
            x = getR();
            printf("sub $%d, $%d, $%d\n", getR(), x, x);
            tot++;
            break;
        case 14:
            printf("subu $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 15:
            printf("mult $%d, $%d\n", getR(), getR());
            tot++;
            break;
        case 16:
            printf("multu $%d, $%d\n", getR(), getR());
            tot++;
            break;
        case 17:
            printf("slt $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 18:
            printf("sltu $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 19:
            printf("sll $%d, $%d, %d\n", getR(), getR(), shift(mt));
            tot++;
            break;
        case 20:
            printf("srl $%d, $%d, %d\n", getR(), getR(), shift(mt));
            tot++;
            break;
        case 21:
            printf("sra $%d, $%d, %d\n", getR(), getR(), shift(mt));
            tot++;
            break;
```

```
        case 22:
            printf("sllv $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 23:
            printf("srlv $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 24:
            printf("srav $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 25:
            printf("and $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 26:
            printf("or $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 27:
            printf("xor $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 28:
            printf("nor $%d, $%d, $%d\n", getR(), getR(), getR());
            tot++;
            break;
        case 29:
            printf("addi $%d, $%d, %d\n", getR(), getR(), 0);
            tot++;
            break;
        case 30:
            printf("addiu $%d, $%d, %d\n", getR(), getR(), s16(mt));
            tot++;
            break;
        case 31:
            printf("andi $%d, $%d, %d\n", getR(), getR(), u16(mt));
            tot++;
            break;
        case 32:
            printf("ori $%d, $%d, %d\n", getR(), getR(), u16(mt));
            tot++;
            break;
        case 33:
            printf("xori $%d, $%d, %d\n", getR(), getR(), u16(mt));
            tot++;
            break;
        case 34:
            printf("lui $%d, %d\n", getR(), u16(mt));
            tot++;
            break;
        case 35:
            printf("slti $%d, $%d, %d\n", getR(), getR(), s16(mt));
            tot++;
            break;
        case 36:
            printf("sltiu $%d, $%d, %d\n", getR(), getR(), s16(mt));
```

```
            tot++;
            break;
        case 37:
            printf("mfhi $%d\n", getR());
            tot++;
            break;
        case 38:
            printf("mflo $%d\n", getR());
            tot++;
            break;
        case 39:
            printf("mthi $%d\n", getR());
            tot++;
            break;
        case 40:
            printf("mtlo $%d\n", getR());
            tot++;
            break;
        case 41:
            printf("beq $%d, $%d, label%d\n", getR(), getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 42:
            printf("bne $%d, $%d, label%d\n", getR(), getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 43:
            printf("blez $%d, label%d\n", getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 44:
            printf("bgtz $%d, label%d\n", getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 45:
            printf("bltz $%d, label%d\n", getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
```

```
        case 46:
            printf("bgez $%d, label%d\n", getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 47:
            printf("j label%d\n", ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 48:
            printf("jal label%d\n", ++cnt);
            X = getR();
            printf("ori $%d, $0, 16\n", X);
            solve(one(mt));
            printf("label%d: addu $%d, $%d, $31\n", cnt, X, X);
            printf("jr $%d\n", X);
            puts("nop");//solve(I(mt));
            tot += 4;
            break;
        case 49:
            printf("jal label%d\n", ++cnt);
            X = getR();
            printf("ori $%d, $0, 16\n", X);
            solve(one(mt));
            printf("label%d: addu $%d, $%d, $31\n", cnt, X, X);
            printf("jalr $%d, $%d\n", getR(), X);
            puts("nop");//solve(I(mt));
            tot += 4;
            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");
    while(tot <100) solve(IJ(mt));
}
```

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。

```
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\P6-2 D:\verilog\P6-2\test.v
vvp tb.out -v > v.out
sort.exe
m_sort.exe
fc aftersort.out maftersort.out
pause
```

三，思考题

（一）为什么需要有单独的乘除法部件而不是整合进ALU？为何需要有独立的hi/lo寄存器？

- 因为乘除法指令需要执行的周期比一般的一些数学运算操作要长很多，如果整合进ALU就会大大提高ex级整体延迟时间，从而大大增加流水线的周期。如果另开一个模块和寄存器，就可以将乘除法的计算同一般的alu计算割裂开来，在不产生矛盾的情况下可以让alu计算和乘除模块并行，从而降低乘除法延迟对流水线周期的影响，保证流水线处理器的效率。

（二）参照你对延迟槽的理解，试解释“乘除槽”。

- 在执行乘除法的时候，会有与乘除法相关的指令被阻塞在if和id级无法被执行，这等同与延迟槽的效果。

（三）举例说明并分析何时按字节访问内存相对于按字访问内存性能上更占优势。

- 当对字节或者半字进行存储的指令居多的情况下，比如lb，sb等指令在指令总数中占了大多数时，按字节访问相对于按字访问会更有优势。

(四) 本实验你遇到了哪些不同的指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

- 比如lw指令和R型指令的组合，两个R型指令的组合，R型指令和sw指令的组合，诸如此类的指令都有可能产生冲突。
- 可以通过转发和暂停来解决。
- 在随机生成测试指令的时候全程只采用3个寄存器，也就是说随机生成的指令永远最多只会用到3个寄存器，这样在多次（成百上千）随即下可以保证冲突的最大性。

(五) 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

- 对于指令进行分类，对相同类型的指令尽量使用同样的数据通路。
- 通过分布式译码的形式，先利用操作码和功能码将具体指令的类型译出，再用或逻辑产生对应数据通路上每一条分支的控制信号，对于即用的信号用的时候再译出，对于全局的信号（比如改寄存器指令的目标寄存器），尽早译出并流水寄存器中传递。
- 规范化命名方式，尽量用名字来表征信号的功能，比如RegWrite（写寄存器），id_ex_to_id（从id_ex向id级转发）。