MIPS微系统设计报告(P7)

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4096条指令，以及IO接口和内核为支持中断异常处理的五级流水线CPU的MIPS微系统，支持MIPS-C4指令集53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），tc1（时钟1），tc2（时钟2），mips（主模块），CPU（流水线处理器），CP0（协处理器）这些模块。

（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
cut	I	中断控制
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
s_wrong	O	存指令溢出
l_wrong	O	取指令溢出

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
IR	I [31:0]	指令码
具体指令译码(略53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
ex_cg_cp0	I	ex级是否是修改cp0寄存器指令
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
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
ex_cgcp0	I	ex级是否是修改cp0的指令
mem_cgcp0	I	mem级是否是修改cp0的指令
wb_cgcp0	I	wb级是否是修改cp0的指令
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_id_cp0	O [1:0]	id级cp0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_ex_cp0	O [1:0]	ex级cp0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

14 TC

A 介绍

模拟外设之一，计时器。

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
Addr	I [31:2]	计时器寄存器代表的地址
WE	I	写入使能端
Din	I [31:0]	写入的数据
Dout	O [31:0]	输出给外界的计时器寄存器的数据
IRQ	O [31:0]	中断信号

C 功能定义

功能名称	功能描述
中断	可以向外部发出中断信号
写读寄存器	完成将数据写入和将数据读出计时器寄存器的操作

15 CP0

A 介绍

协处理器，帮助CPU处理异常与中断。

B 端口定义

端口	输出输出（位宽）	描述
clk	I	时钟信号
reset	I	清空信号
IR	I [31:0]	id级的指令
WA	I [4:0]	修改寄存器编号
is_delay	I	判断当前mem级指令是否为延迟槽指令
is_eret	I	判断当前wb级指令是否为eret指令
WE	I	写入CP0寄存器使能信号
WD	I [31:0]	写入CP0寄存器的数据
PC	I [31:0]	错误指令的PC
exccode	I [4:0]	异常代码
HWInt	I [5:0]	外部中断信号
cut	O	中断信号
outdata	O [31:0]	cp0输出的数据

C 功能定义

功能名称	功能描述
处理中断	产生中断信号，以及在中断来临时修改CP0寄存器
读写数据	完成将数据写入和将数据读出CP0寄存器的操作

16 CPU

A 介绍

MIPS微系统中处理器的部分

B 端口定义

端口	输出输出（位宽）	描述
clk	I	时钟信号
reset	I	清空信号
PrRD	I [31:0]	从外部读入的数据
HWInt	I [5:0]	6个外部硬件中断
Praddr	O [31:2]	地址总线
PrWD	O [31:0]	向外部传输的数据
PrWrite	O	向外部写使能
out_PC	O [31:0]	宏观PC

C 功能定义

功能名称	功能描述
沟通外部	与外部实现数据交换
接受中断	接收来自外部的中断信号

17 mips

A 介绍

微处理器主模块，负责CPU和各外设的协调。

B 端口定义

端口	输出输出（位宽）	描述
clk	I	时钟信号
reset	I	清空信号
interrupt	I	中断信号
addr	O [31:0]	输出宏观PC

C 功能定义

功能名称	功能描述
协调数据传输	构造桥作为数据通路，协调CPU和外设的数据传输

（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								

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）。

4 桥

不另设模块，利用tc1和tc2的地址特点与从CPU模块中传输过来的数据，在mips主模块利用assign语句用三目运算符进行条件判断之后赋值。

```
assign tc1_WE = PrWrite && ({Praddr,2'b00} <= 32'h00007f0b);
assign tc2_WE = PrWrite && ({Praddr,2'b00} >= 32'h00007f10);
assign HWInt = {3'b000,interrupt,tc2_IQR,tc1_IQR};
assign T_din = ({Praddr,2'b00} <= 32'h00007f0b) ? tc1_out : tc2_out;
```

5 异常中断及CP0处理

关于CP0

对于CP0，我在五级流水线中将它看作和grf同等地位，对于CP0的读写操作，与grf相同，读的时候直接取值，写的时候进行回写。同时我增加了对于CP0相关寄存器的指令的转发操作。当出现异常和中断时，CP0内部优先相应wb级的回写操作，之后进行中断时的修改cp0寄存器的操作。

```
if(WE) begin
    if(WA == 12) begin
    if(is_eret) SR[1] <= 0;
    else SR <= WD;
    end
    else if(WA == 13) cause <= WD;
    else if(WA == 14) EPC <= WD;
    else if(WA == 15) PRID <= WD;
end
if(cut) begin
    SR[1] <= 1'b1;
    if(is_delay) cause[31] <= 1'b1;
    else cause[31] <= 1'b0;
    cause[6:2] <= out_cut ? 0 : exccode;
    EPC <= (is_delay ? {PC[31:2],2'b00}-4 : {PC[31:2],2'b00});
end
  cause[15:10] <= HWInt;
end
```

关于外部中断

对于外部中断，cp0中统一判断mem级指令的异常与中断，将6个外部中断信号与6个中断使能位进行与操作，同时将结果与上外部中断使能与exl信号。

```
assign out_cut = ((_SR[15:10] & HWInt)&& _SR[0] && !_SR[1]);//外部中断条件
```

关于异常

对于异常，我采取了分级判断的手段。在if级判断PC异常，在id级判断未知指令，在ex级判断算数和取址计算溢出，在mem级判断存数和取数异常，并将异常码（exccode）在流水线中流水，最终在mem级将最终处理好的exccode传入cp0，交给cp0进行处理。

二，测试方案

自动化生成测试数据

仅仅适用于除eret外，且不包含中断异常的指令执行正确性测试。

其实下面数据生成器的是测不了异常中断的……

c++编程生成测试数据：

```
#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, 42),
    J(43, 51),
    IJ(1, 51),
    one(11, 42),
    cp0(12, 14);

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("mfc0 $%d, $%d\n", getR(), cp0(mt));
            tot++;
            break;
        case 42:
            x = cp0(mt);
            printf("mtc0 $%d, $%d\n", getR(), x == 13 ? 12 : x);
            tot++;
            break;
        case 43:
            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 44:
            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 45:
            printf("blez $%d, label%d\n", getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
```

```
            break;
        case 46:
            printf("bgtz $%d, label%d\n", getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 47:
            printf("bltz $%d, label%d\n", getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 48:
            printf("bgez $%d, label%d\n", getR(), ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 49:
            printf("j label%d\n", ++cnt);
            solve(I(mt));
            solve(I(mt));
            printf("label%d: ", cnt);
            solve(I(mt));
            tot++;
            break;
        case 50:
            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 51:
            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");
    puts("mtc0 $0, $12");
    while(tot < 900) 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
```

其他涉及中断和异常以及桥的测试代码都是嫖的别的同学或是往届学长的（

三，思考题

（一）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”（Tips:什么是接口？和我们到现在为止所学的有什么联系？）？

- "硬件/软件接口"是一个指令集体系。一个指令集体系规定了一个硬件系统所能实现的所有功能。软件做到将指令集中的指令集成化，并再次抽象化，使其可以实现更加复杂的操作。而指令集中的指令通过一定的规约赋予二进制代码涵义并交给硬件电路，硬件电路将这些逻辑转化为物理层面的操作做出相应的回应，从而实现目标的功能。所以，机器码起到了软件和硬件沟通的作用。

（二）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

- DM应该处于CPU的外部，因为现代计算机有着处理速度块，存储容量大的特点，需要更多的外设来提高计算机的性能，比如cache和虚拟存储器等。DM独立于CPU有利于更加方便的实现这些外设的功能。

（三）BE部件对所有的外设都是必要的吗？

- 不是的，只有对那些涉及到字节或半字存取操作的外设才有必要。

（四）请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别对每种模式绘制状态转换图

-
    - 模式0：在计时结束后触发中断，在使能端重新置1之前定时器会产生连续中断信号。
    - 模式1：在计时结束后触发一周期中断信号，再次进入置位计数循环中。
- 见定时器说明文档

(五) 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：

- 定时器在主程序中被初始化为模式0。
- 定时器倒计数至0产生中断。
- handler设置使能Enable为1从而再次启动定时器的计数器。2及3被无限重复。
- 主程序在初始化时将定时器初始化为模式0，设定初值寄存器的初值为某个值，如100或1000。（注意，主程序可能需要涉及对CP0.SR的编程，推荐阅读过后文后再进行。）

```
.text
ori $5 $5 0xfc01
mtc0 $12 $5
ori $6 $6 9
ori $7 $7 100
sw $6 0x7f00($0)
sw $7 0x7f04($0)
label:
j label

.ktext 0x4180
ori $6 $6 9
ori $7 $7 100
sw $6 0x7f00($0)
sw $7 0x7f04($0)
eret
```

(六) 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

- 鼠标和键盘的输入信号通过操作系统传输给CPU，引起CPU触发中断，在对应中断处理程序中，将鼠标键盘改变的值存入对应的寄存器，之后跳出中断，完成一次修改。