

计算机系统结构课程实验

总结报告

实验题目：动态流水线设计与性能定量分析

学号：1953072

姓名：肖鹏飞

指导教师：陆有军

日期：2021.12.11

1. **实验环境部署与硬件配置说明**

1.实验环境

实验软件：vivado 2016.2

仿真测试软件：ModelSim PE 10.4c

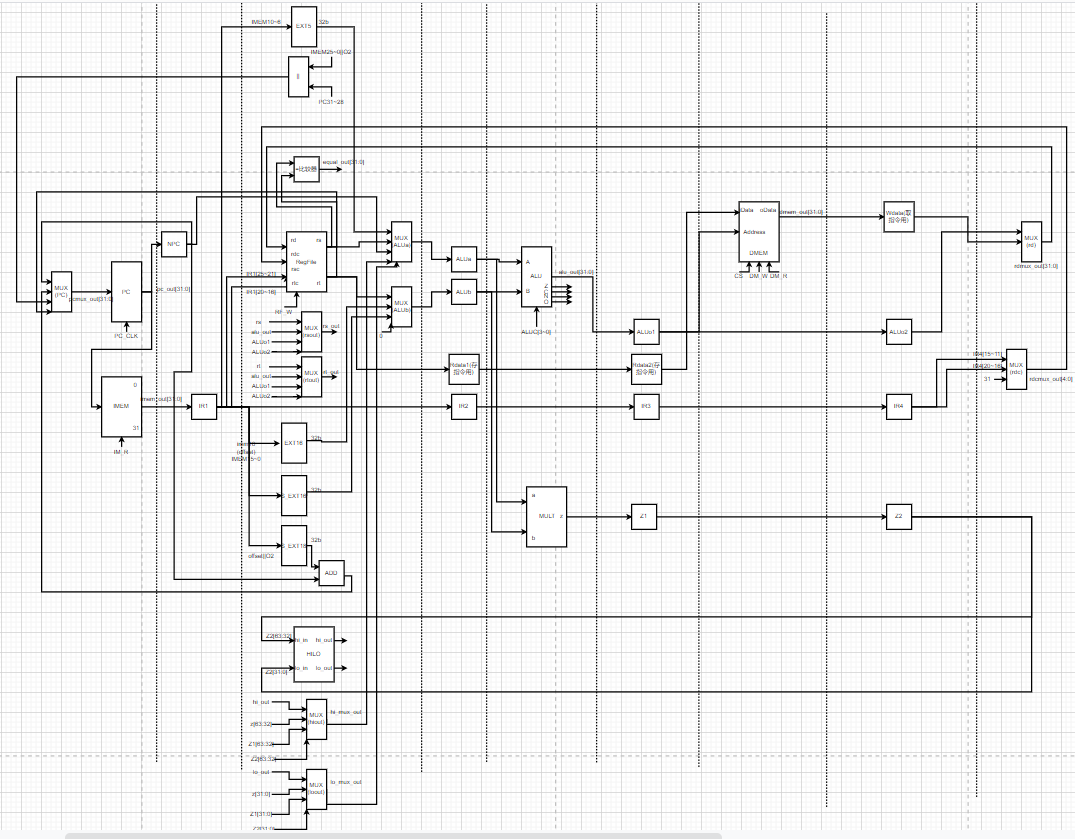
2.硬件配置

开发板：XILINX Artix-7 NEXYS 4 DDR

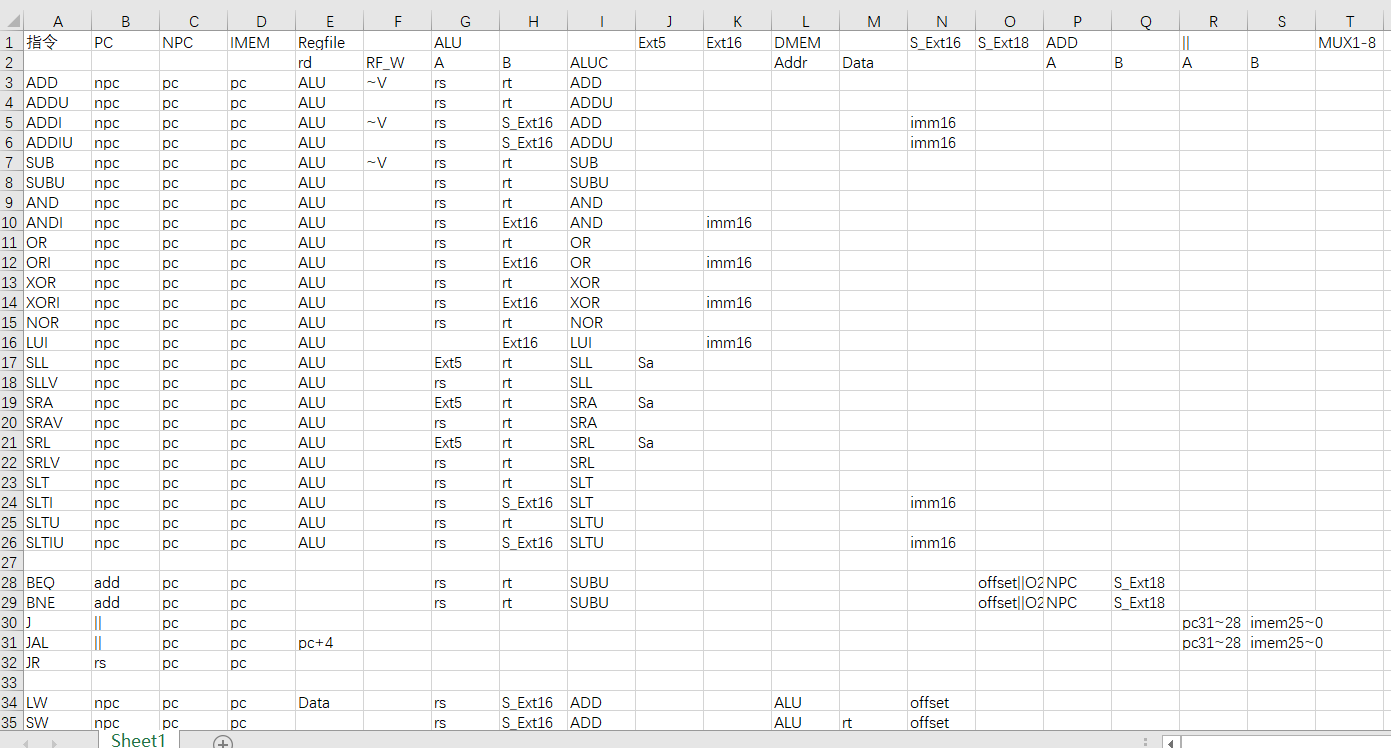
1. **实验的总体结构**

1.动态流水线的总体结构

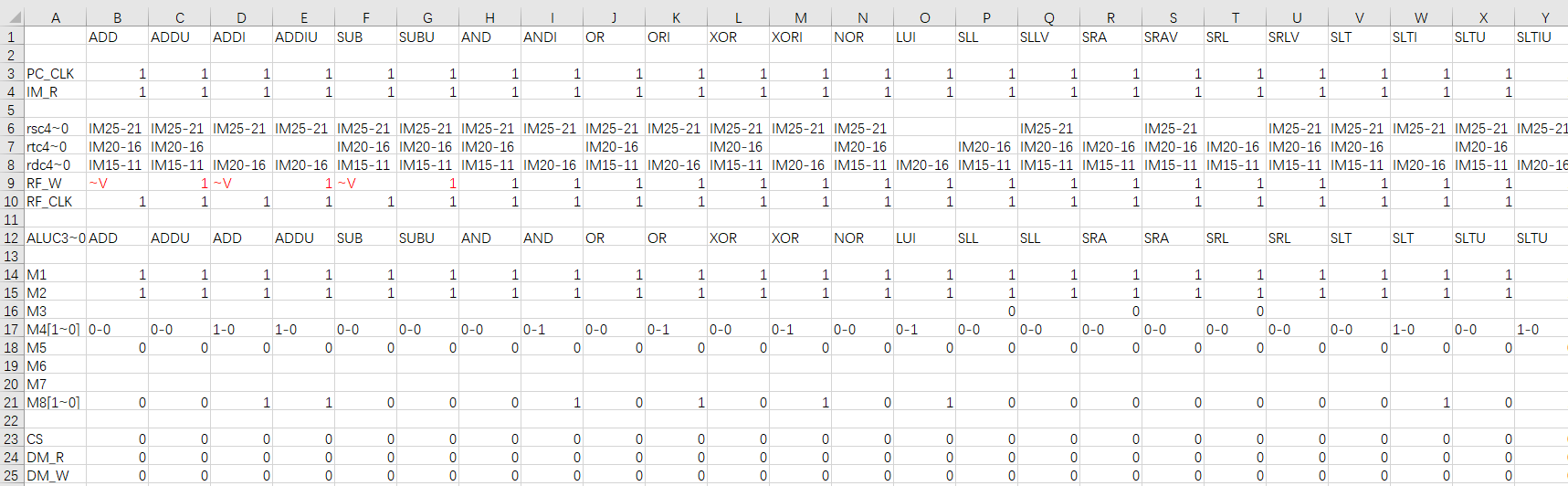
CPU总体结构（按流水段分解）：

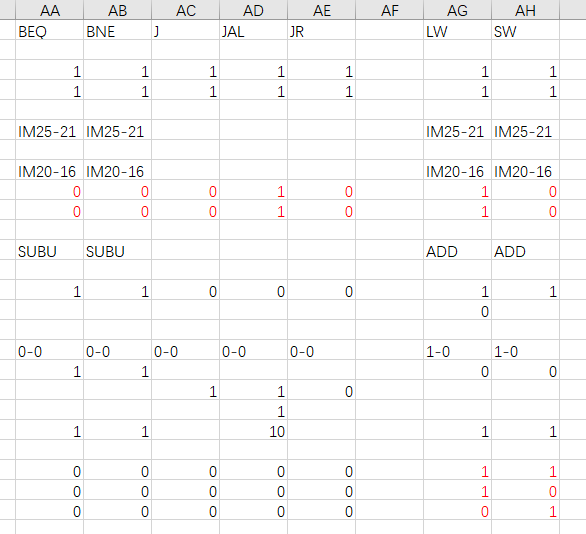


部件接口连接图：



控制信号：





1. **总体架构部件的解释说明**

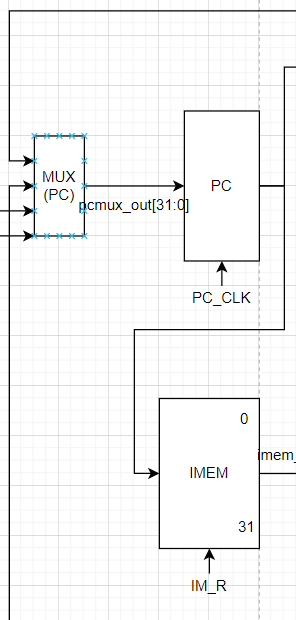
1.动态流水线总体结构部件的解释说明

(1).IF级部件

输入：PC来源，clk、reset等控制信号。

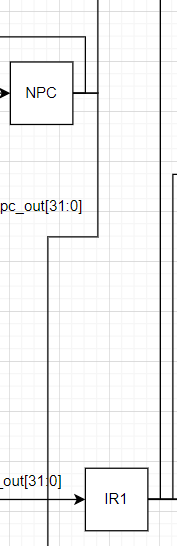
输出：和当前PC对应的指令及下一条PC。

作用：根据PC选择指令。



(2).IF-ID级段间流水部件

作用：流水存储取出的指令及PC+4。

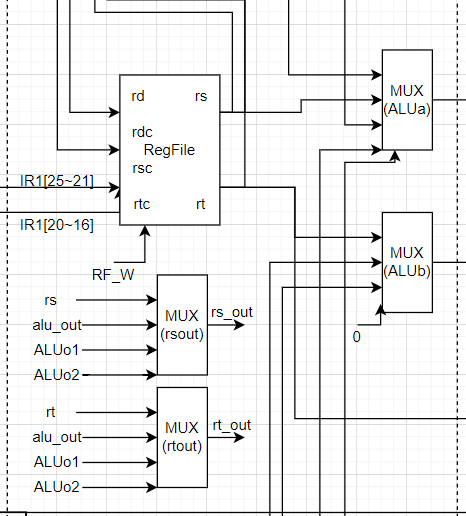


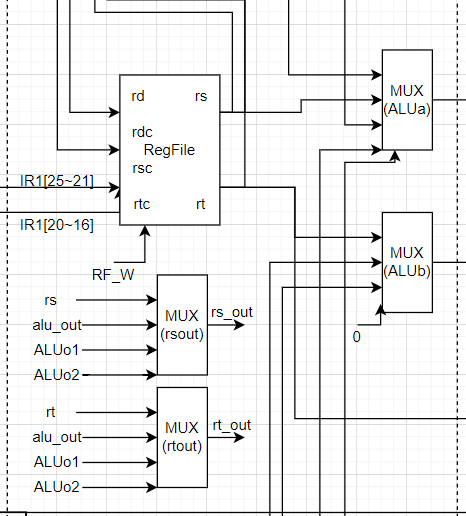
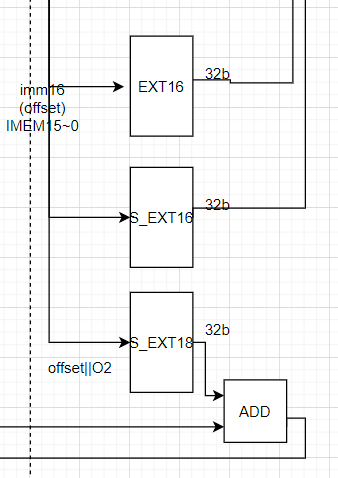
(3).ID级部件

输入：上一个流水段的指令，需要输入的寄存器地址和值，及各个部件需要的控制信号。

输出： 扩展后的数据，读出的rs和rt内容。

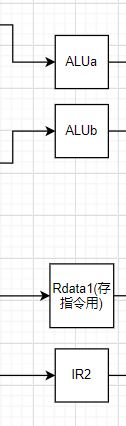
作用：数据扩展，转移指令的比较和计算，寄存器堆的读写，HI、LO寄存器的读写。





(4).ID-EX级段间流水部件

作用：流水存储rs、rt等寄存器的值及之后流水段的所需指令。

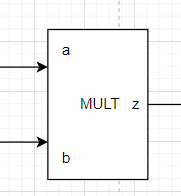
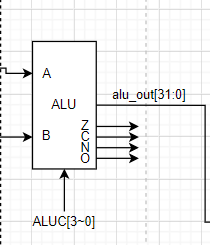


(5).EX级部件

输入：ALU的两个输入及需要的控制信号。

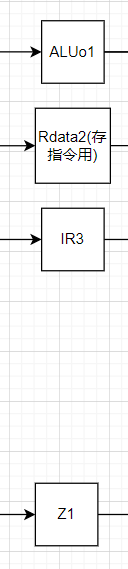
输出：ALU计算得到的结果。

作用：完成计算功能。



(6).EX-ME级段间流水部件

作用：流水存储ALU的计算结果及之后流水段的所需指令。

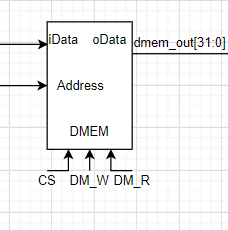


(7).ME级部件

输入：数据存储器的地址和数据及需要的控制信号。

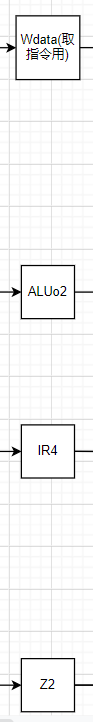
输出：从数据存储器中取出的值。

作用：访问存储器，完成存取功能。



(8).ME-WB级段间流水部件

作用：流水存储DMEM的取出内容及之后流水段的所需指令。

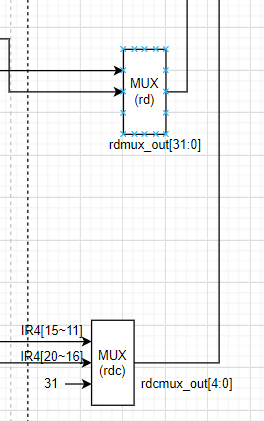


(9).WB级部件

输入：从DMEM取出的数据和ALU计算出的数据及需要的控制信号。

输出：寄存器堆的输入地址和数据。

作用：将数据写回寄存器堆。

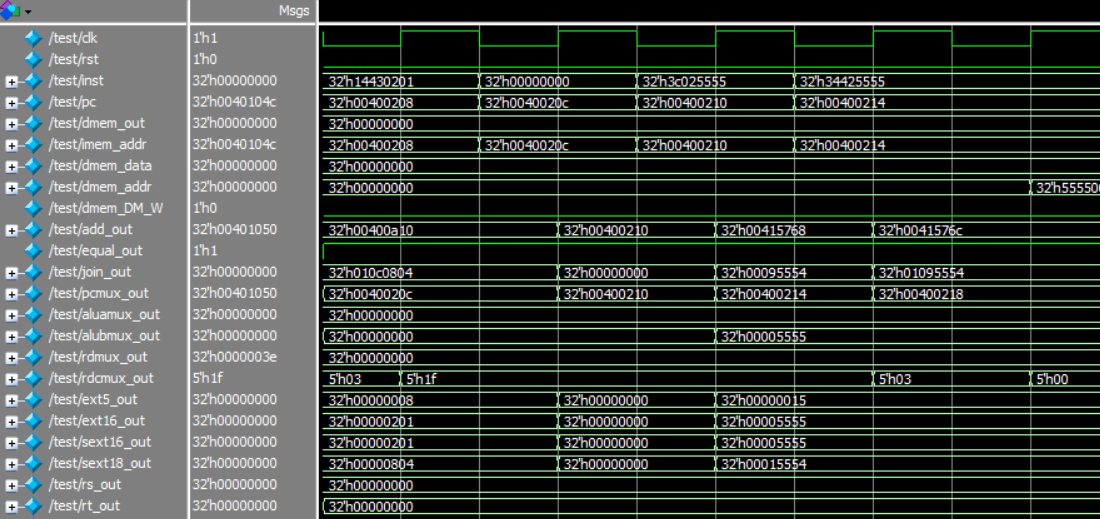


1. **实验仿真过程**

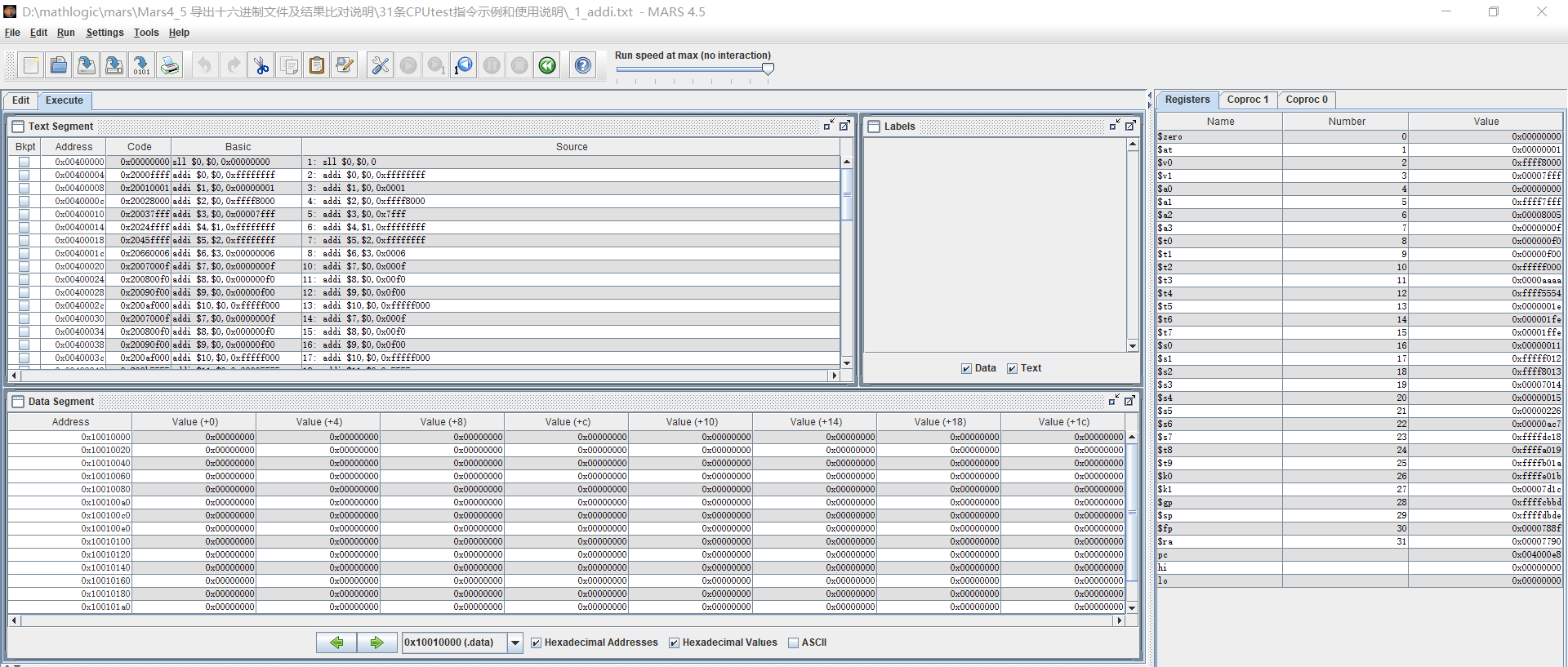
1.动态流水线的仿真过程

(1).前仿真过程

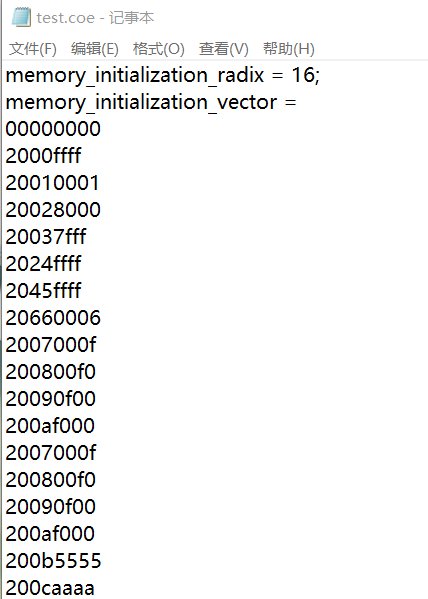
ModelSim波形图：



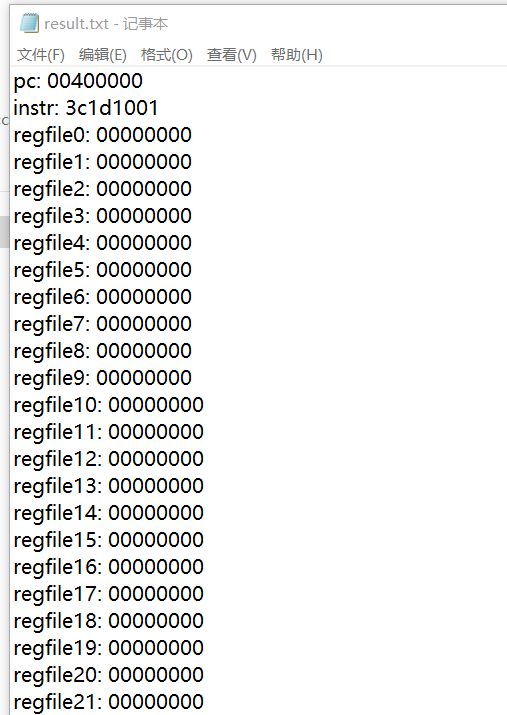
从Mars中获取coe文件和result文件：



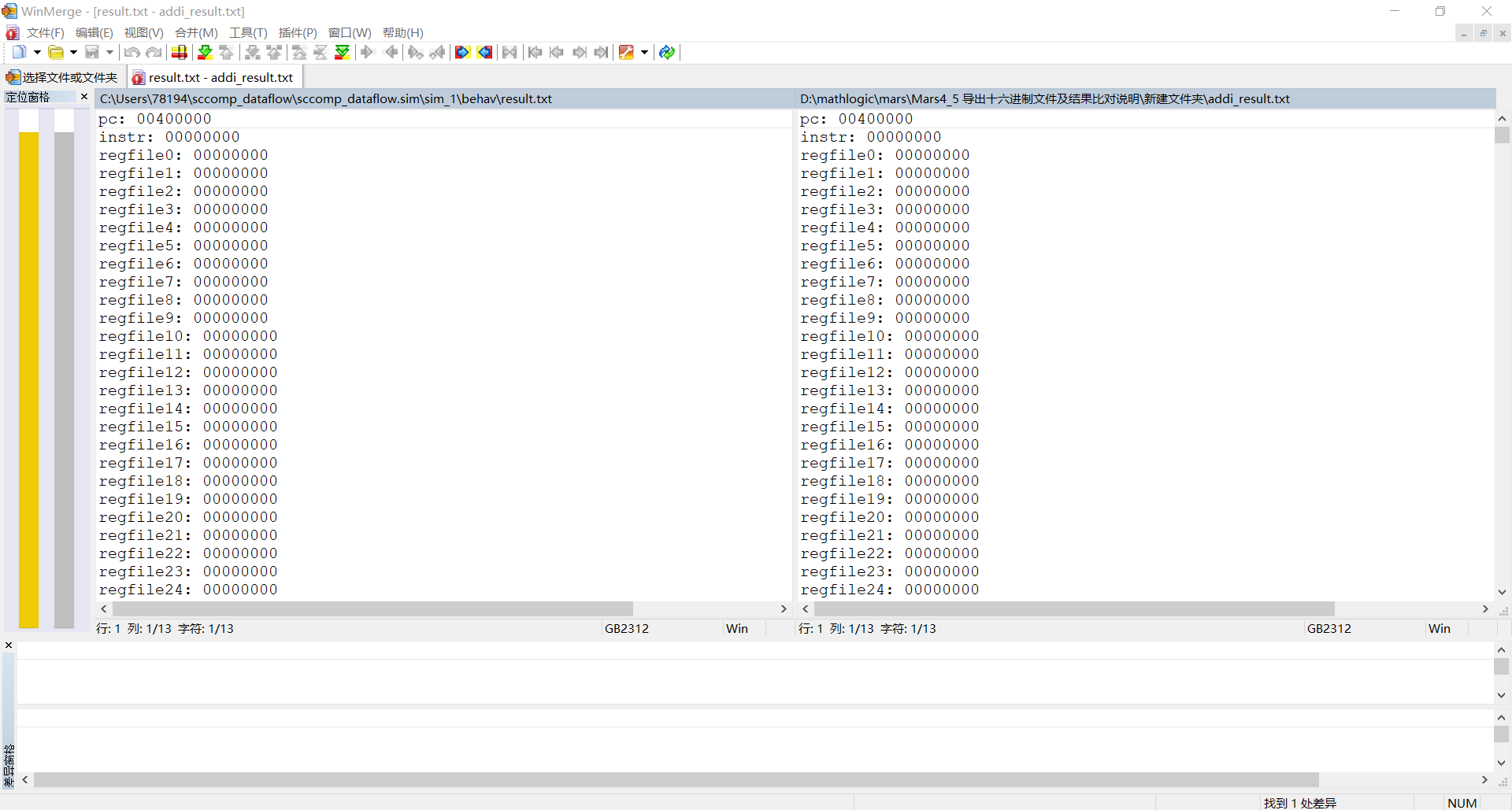
Mars导出的coe文件：



过程中打印的result.txt文件：



生成的result文件与Mars的执行结果在文本比较器中比较：

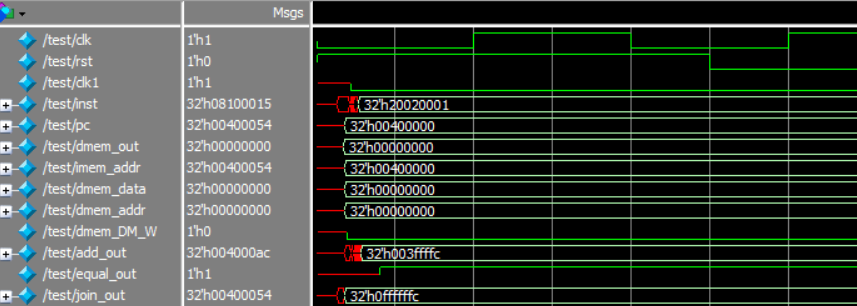


(2).后仿真

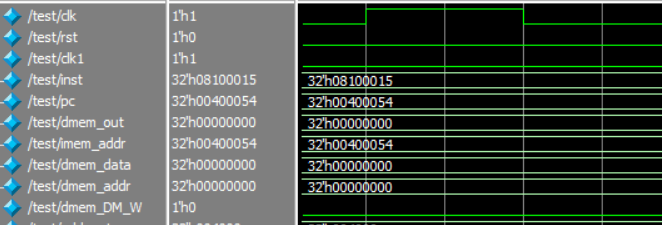
后仿真无法在testbench中打印寄存器的值与result比较，因此只能观察波形图。

图中的clk1为200倍分频之后的时钟。

起始时刻，存在数据通路延迟：



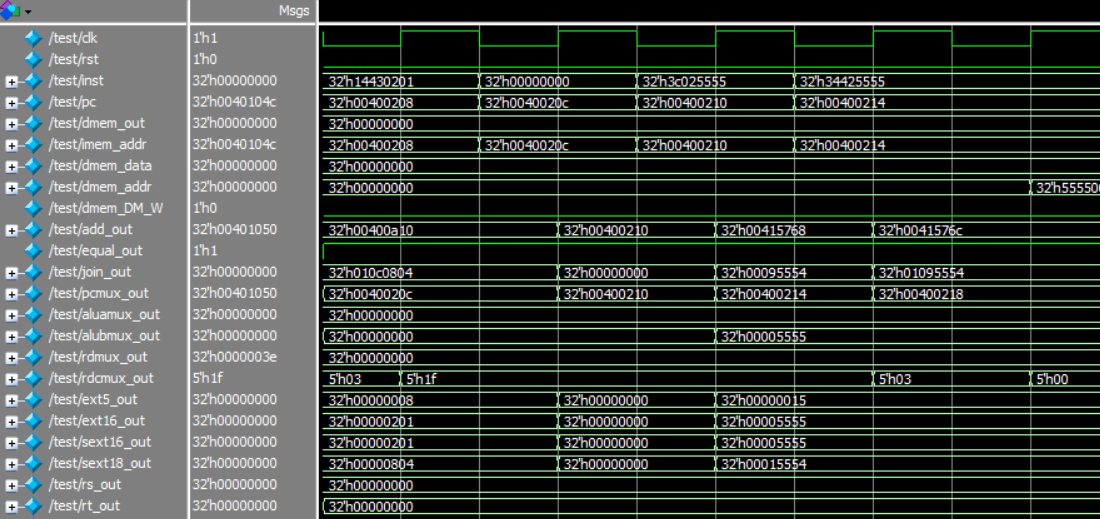
程序执行完成，各项数据稳定：

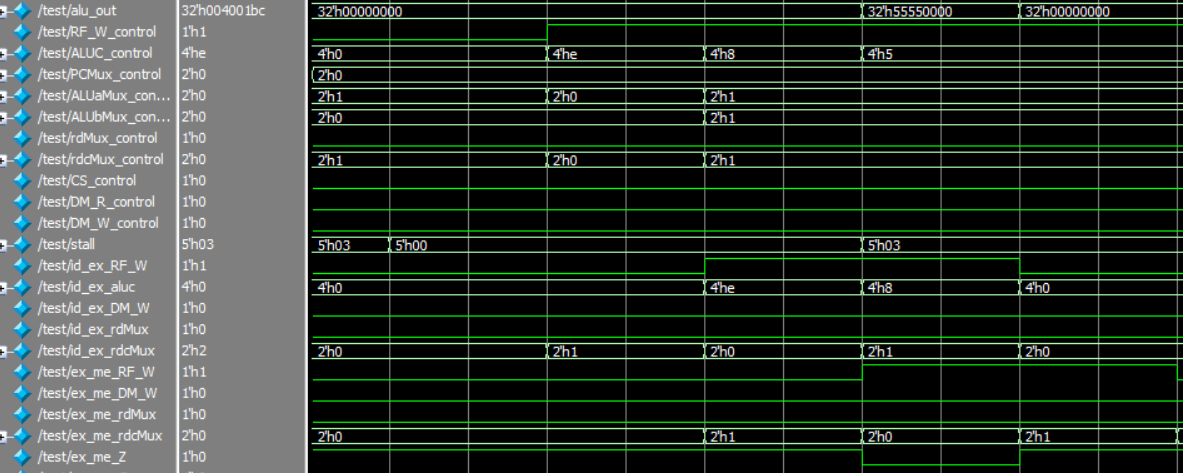


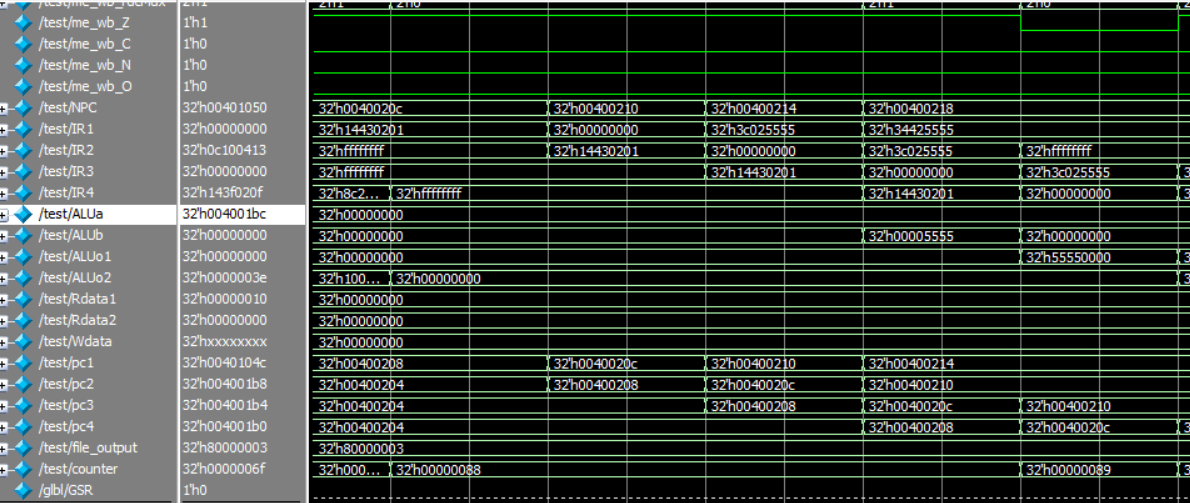
1. **实验仿真的波形图及某时刻寄存器值的物理意义**

1.动态流水线的波形图及某时刻寄存器值的物理意义

全局输出的信号量清单：







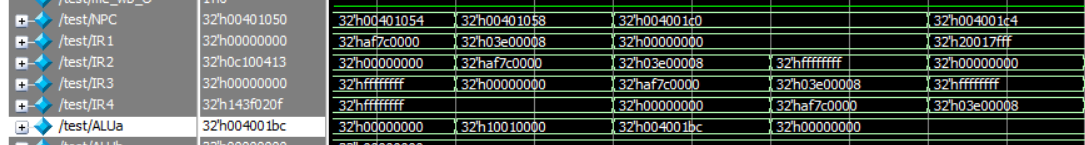
1.正常执行

横向看，分别是每一级的信号输出，纵向看，是每个时钟周期内每一级的当前输出。在下图中，指令在流水线中顺次流动，按照IF-IF-EX-ME-WB的顺序流动，并将每一级的输出和信号量传递到下一级。

PC与指令同步情况：



指令流水线情况：



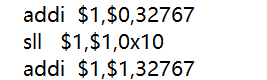
2.遇到数据冲突的暂停情况

横向看，分别是每一级的信号输出，纵向看，是每个时钟周期内每一级的当前输出。

解决数据冲突的方式根据前后的指令不同选择不同的方式。对于ALU相关的计算指令（ADDI等），使用数据前推来消除冲突。采用专用路径的方式将ex、me、wb段中未写入的内容送入id段作为读取内容。而对于内存操作（LW）、数据转移（JAL）等指令，则采用插入空指令的方式进行数据的推迟。

（1）.数据前推方法，专用路径，流水线不暂停

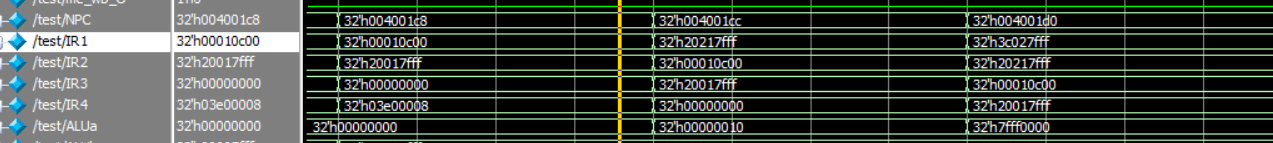
汇编指令，数据冲突：



PC与指令同步情况：



实际：流水线未暂停，正常流动：



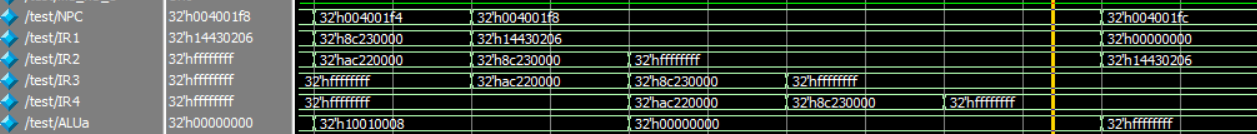
（2）.插入stall延迟情况

在下图中，指令发生数据冲突，流水线前半部分停止流动，在两者之间不断插入空指令，直到不再发生数据相关时继续流动。

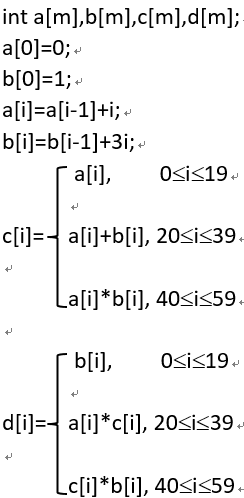
PC与指令同步情况：



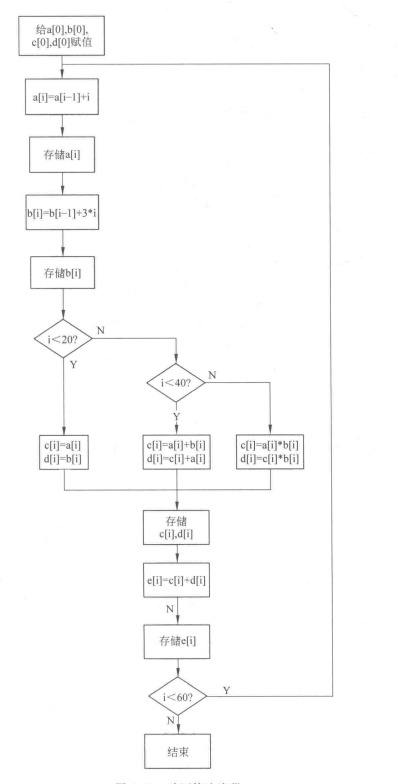
指令流水线情况：



1. **实验验算数学模型及算法程序**



算法流程图：



算法汇编代码：

.data

A:.space 240

B:.space 240

C:.space 240

D:.space 240

E:.space 240

.text

j main

exc:

nop

j exc

main:

addi $2,$0,0

addi $3,$0,1

addi $4,$0,0

addi $13,$0,0

addi $5,$0,4

addi $6,$0,0

addi $7,$0,1

addi $10,$0,0

addi $11,$0,240

addi $14,$0,3

addi $30,$0,0

lui $27,0x0000

addu $27,$27,$0

sw $2,A($27)

lui $27,0x0000

addu $27,$27,$0

sw $3,B($27)

lui $27,0x0000

addu $27,$27,$0

sw $2,C($27)

lui $27,0x0000

addu $27,$27,$0

sw $3,D($27)

loop:

srl $12,$5,2

add $6,$6,$12

lui $27,0x0000

addu $27,$27,$5

sw $6,A($27)

mult $14,$12

mflo $15

add $7,$7,$15

lui $27,0x0000

addu $27,$27,$5

sw $7,B($27)

slti $10,$5,80

bne $10,1,c1

lui $27,0x0000

addu $27,$27,$5

sw $7,D($27)

addi $15,$6,0

addi $16,$7,0

j endc

c1:

slti $10,$5,160

addi $27,$0,1

bne $10,$27,c2

add $15,$6,$7

lui $27,0x0000

addu $27,$27,$5

sw $15,C($27)

mult $15,$6

mflo $16

lui $27,0x0000

addu $27,$27,$5

sw $16,D($27)

j endc

c2:

mult $6,$7

mflo $15

lui $27,0x0000

addu $27,$27,$5

sw $15,C($27)

mult $15,$7

mflo $16

lui $27,0x0000

addu $27,$27,$5

sw $16,D($27)

endc:

add $28,$15,$16

lui $27,0x0000

addu $27,$27,$5

sw $28,E($27)

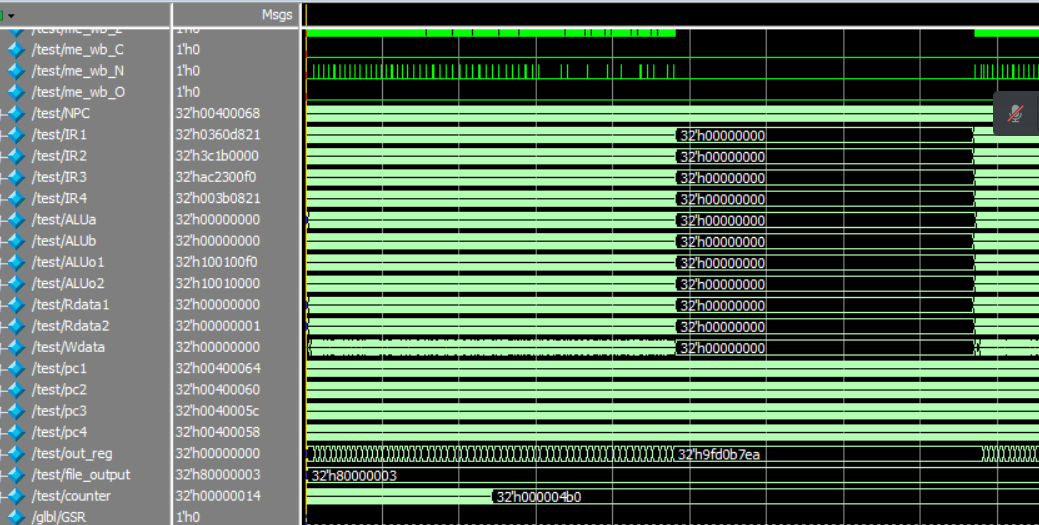
addi $5,$5,4

bne $5,$11,loop

在MARS软件中得到的预期结果：0x9fd0b7ea



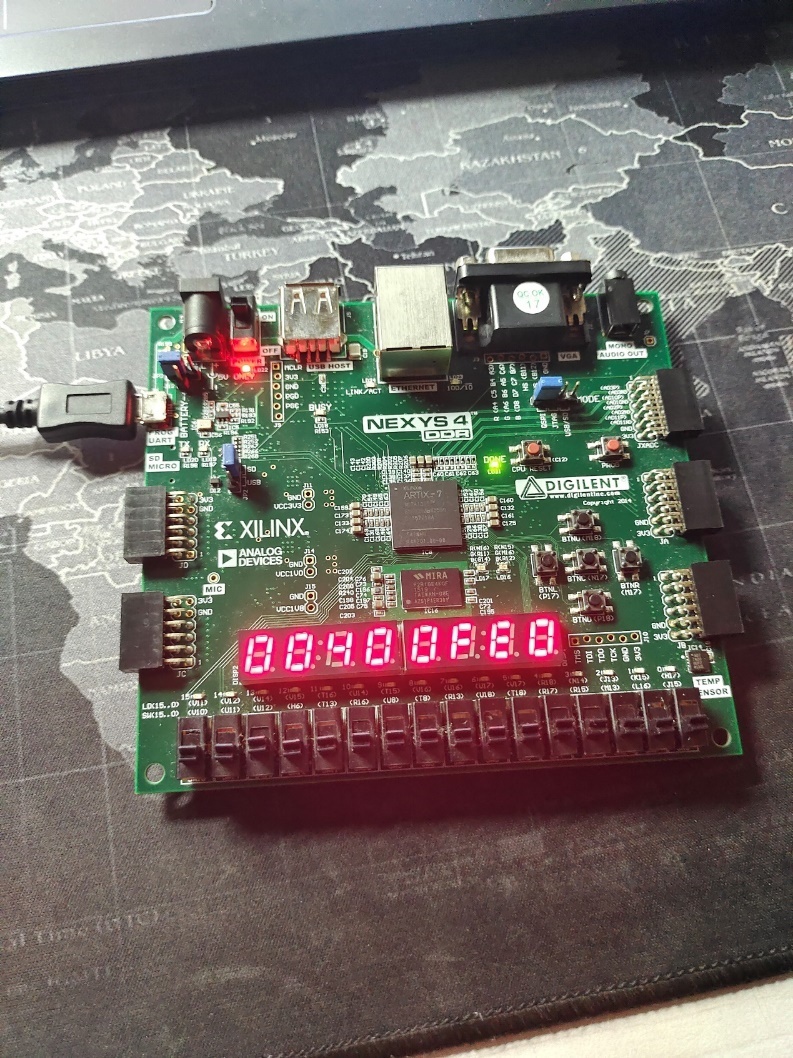
仿真得到的结果（前仿真，后仿真结果相同）：0x9fd0b7ea



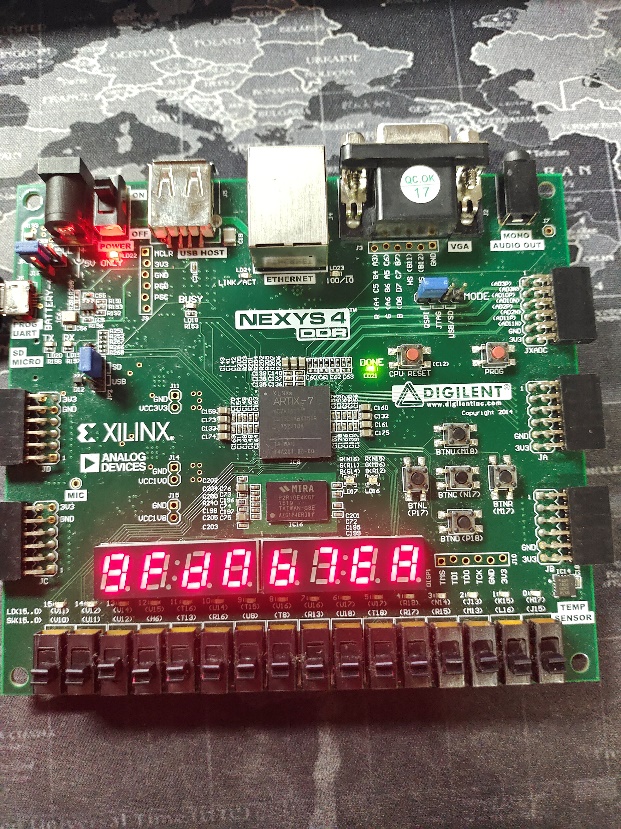
由以上两图可见仿真得到的结果符合预期，动态流水线在仿真阶段正确运行。

1. **实验验算程序下板测试过程与实现**

CPU测试pc值：

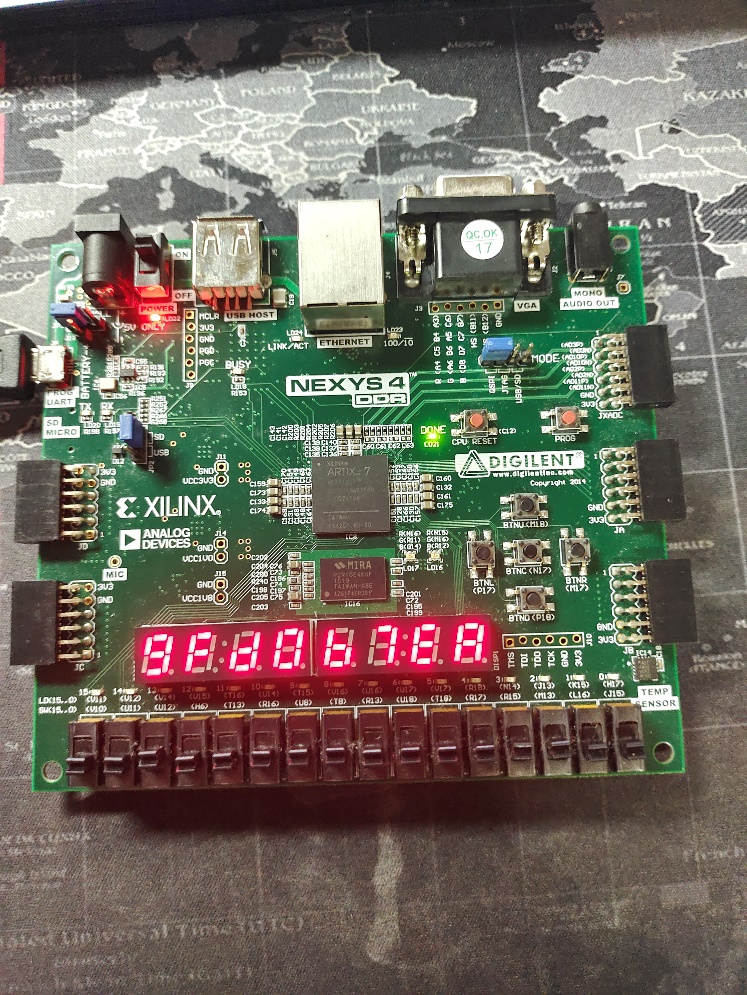


下板过程中的数组内容：

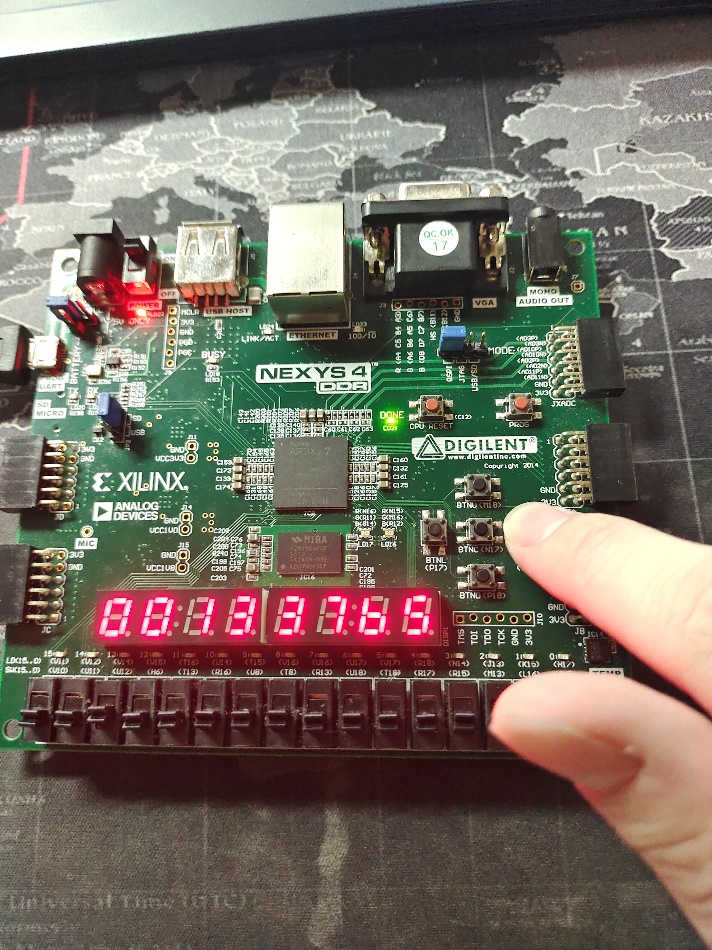




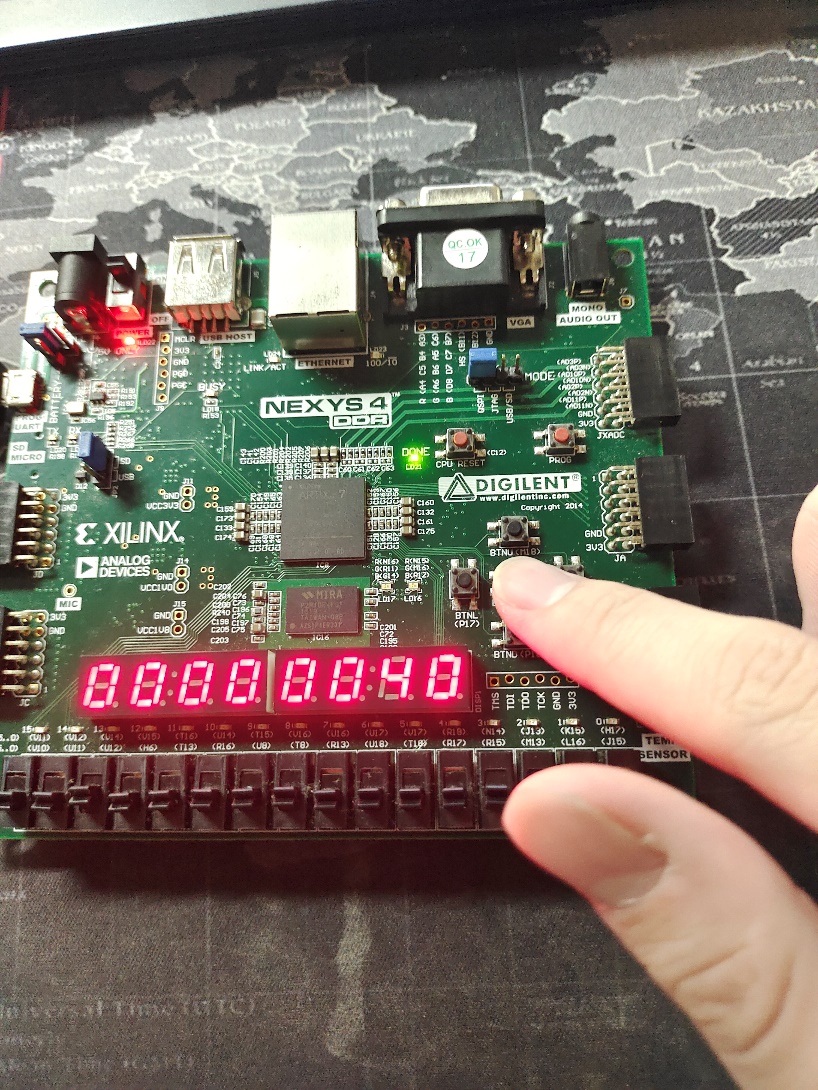
稳定在最后一个数组内容：



中断stop：



重置reset：



由上图可知下板后得到的实验结果和（六）中MARS得到的结果和仿真得到的结果相同，均为0x9fd0b7ea，因此可以证明下板结果是正确的。

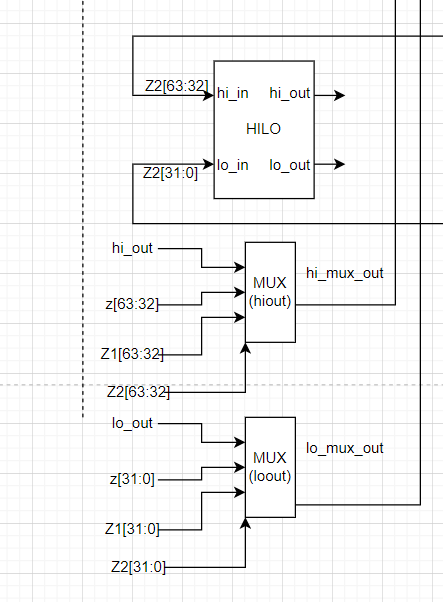
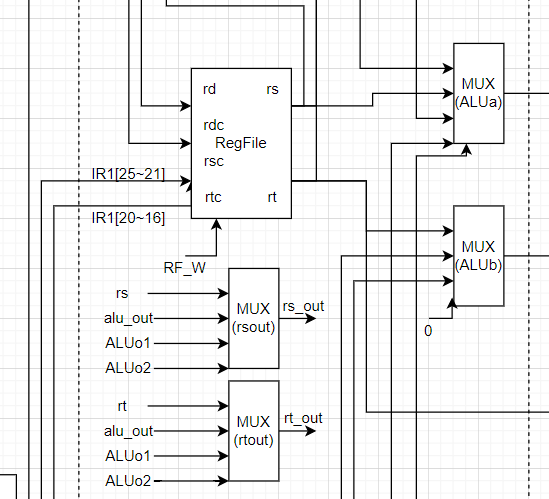
1. **流水线的性能指标定性分析（包括：吞吐率、加速比、效率及相关与冲突分析）**

一．对动态流水线性能指标的定性分析：

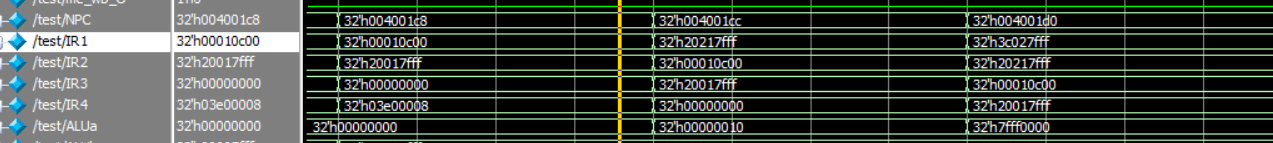
1.数据相关

多条指令数据相关：根据指令的不同选择不同的解决方案。

（1）.采用数据前推的方法，使用专用路径来解决数据相关问题。绝大部分指令(ALU)在第三个周期(EX)就已经得出了结果，而需要取数据的一般只在第二个周期(ID)。因此，对ID-EX,ID-ME,ID-WB三个流水段进行专用路径的设置，应用于HILO寄存器和regfile寄存器堆上，具体实现如下：

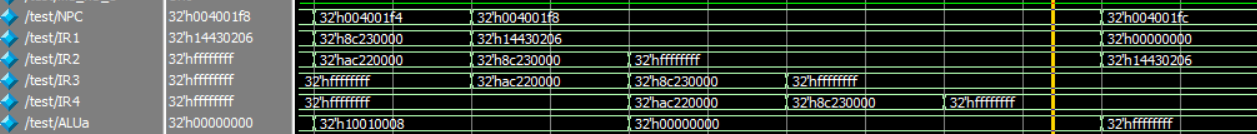


数据前推，流水线不暂停：



（2）.采用暂停流水线（在段间流水寄存器中插入空指令）的方法来解决数据相关问题。读寄存器出现在ID段，写寄存器出现在WB段，因此需要检测的数据冲突为ID-EX,ID-ME,ID-WB三个位置。这个方法只应用于跳转指令和内存读取指令。

由于数据相关而插入空指令的实例：



2.控制相关

对于J、JR、JAL、BEQ、BNE这几条分支和跳转指令，采用延迟槽的思想，并将其判断分支前移至ID级，使得分支延迟为一个周期。

当分支成功或者转移发生时，令处于延迟槽内的指令失效。

当分支失败或转移未发生时，延迟槽内的指令正常执行。

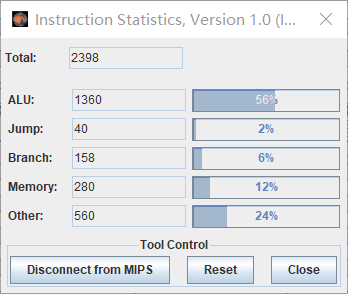
3.冲突

(1).数据冲突：利用数据前推和流水线暂停消除冲突。

(2).控制冲突：利用延迟槽技术消除冲突。

二．对动态流水线性能指标的定量分析：

MARS软件对（七）中汇编程序的统计结果如下：



指令总数：2398条。

ALU指令：1360条，占比56%。

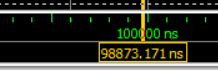
跳转指令：40条，占比2%。

分支指令：158条，占比6%。

内存读写指令：280条，占比12%。

其他指令：560条，占比24%。

寄存器中数据达到稳定的时间点：



仿真得出结果总共用了98800ns，98800/40=2472时钟周期。

1．吞吐率

完成任务总共执行了2398条指令，耗费2472个时钟周期。因此计算所得的吞吐率为：

2.加速比

顺序完成2398条指令所需要的时间为：(1360+40+560)\*5+280\*4+158\*2=11236个时钟周期。实际用时为2472个时钟周期，所以加速比为：

3.效率

完成任务总共执行了2398条指令，指令顺序执行的周期数在上式中已经计算，为11236，因此得到：

1. **总结与体会**

在本次计算机系统结构实验中，我完成了31+3条指令（包括mult、mfhi、mflo在内的三条额外指令）的动态流水线CPU设计、分析与验证。在设计方面，我按照流水段划分构建了动态流水线的总体结构，并具体列出了部件接口和控制信号。具体到每一个流水段的部件、输入、输出、功能都进行了列举分析。在CPU验证方面，我依次完成了前仿真，后仿真和具体的下板实验，通过测试动态流水线CPU所支持的coe，观察波形图并与标准输出文件相比较，从而验证CPU的功能正确实现。

同时，我直接使用汇编语言设计循环相乘求数组中内容的算法，同时利用MARS编译器生成MIPS指令集下的可执行目标程序（coe），并且成功进行了下板验证。最后，我对流水线的性能指标进行了定性分析，包括吞吐率、加速比、效率及相关与冲突分析、CPU的运行时间及存储器空间的使用等性能参数。

在实验的全过程中，我提高了自己对于动态流水线CPU具体内容和实现方法的认识，通过亲自实践一个完整的动态流水线CPU设计、分析与验证过程，巩固了在课上所学到的知识，增进了对系统结构相关内容的了解。除此以外，我对硬件电路设计、仿真流程、DEBUG工具的使用等方面也有了更进一步的了解。

1. **附件（所有程序）**

1.动态流水线的设计程序

module pc(

input PC\_CLK,

input reset,

input wena,

input [31:0] pc\_in,

output reg [31:0] pc\_out

);

always @(negedge PC\_CLK or posedge reset)//上升沿 or 下降沿？

begin

if(reset==1'b1)

pc\_out<=32'h00400000;

else if(wena==1)

pc\_out<=pc\_in;

else

begin

//pc\_out<=pc\_in;

end

end

endmodule

module Regfiles(

input clk,

input rst,

input rf\_w,

input [4:0]raddr1,

input [4:0]raddr2,

input [4:0]waddr,

input [31:0]wdata,

output reg [31:0]rdata1,

output reg [31:0]rdata2,

output reg [31:0] outreg1,

output reg [31:0] outreg2

);

reg [31:0] array\_reg [31:0];//reg数组,前面为位数，后面为数组元素个数

always @(\*)

begin

outreg1=array\_reg[11];

outreg2=array\_reg[12];

end

always @(negedge clk or posedge rst)

begin

if(rst==1'b1)

begin

array\_reg[0]=32'b0;

array\_reg[1]=32'b0;

array\_reg[2]=32'b0;

array\_reg[3]=32'b0;

array\_reg[4]=32'b0;

array\_reg[5]=32'b0;

array\_reg[6]=32'b0;

array\_reg[7]=32'b0;

array\_reg[8]=32'b0;

array\_reg[9]=32'b0;

array\_reg[10]=32'b0;

array\_reg[11]=32'b0;

array\_reg[12]=32'b0;

array\_reg[13]=32'b0;

array\_reg[14]=32'b0;

array\_reg[15]=32'b0;

array\_reg[16]=32'b0;

array\_reg[17]=32'b0;

array\_reg[18]=32'b0;

array\_reg[19]=32'b0;

array\_reg[20]=32'b0;

array\_reg[21]=32'b0;

array\_reg[22]=32'b0;

array\_reg[23]=32'b0;

array\_reg[24]=32'b0;

array\_reg[25]=32'b0;

array\_reg[26]=32'b0;

array\_reg[27]=32'b0;

array\_reg[28]=32'b0;

array\_reg[29]=32'b0;

array\_reg[30]=32'b0;

array\_reg[31]=32'b0;

end

else

begin

if(rf\_w==1)

begin

if (waddr == 0)

array\_reg[0] = 0;

else if (waddr == 1)

array\_reg[1] = wdata;

else if (waddr == 2)

array\_reg[2] = wdata;

else if (waddr == 3)

array\_reg[3] = wdata;

else if (waddr == 4)

array\_reg[4] = wdata;

else if (waddr == 5)

array\_reg[5] = wdata;

else if (waddr == 6)

array\_reg[6] = wdata;

else if (waddr == 7)

array\_reg[7] = wdata;

else if (waddr == 8)

array\_reg[8] = wdata;

else if (waddr == 9)

array\_reg[9] = wdata;

else if (waddr == 10)

array\_reg[10] = wdata;

else if (waddr == 11)

array\_reg[11] = wdata;

else if (waddr == 12)

array\_reg[12] = wdata;

else if (waddr == 13)

array\_reg[13] = wdata;

else if (waddr == 14)

array\_reg[14] = wdata;

else if (waddr == 15)

array\_reg[15] = wdata;

else if (waddr == 16)

array\_reg[16] = wdata;

else if (waddr == 17)

array\_reg[17] = wdata;

else if (waddr == 18)

array\_reg[18] = wdata;

else if (waddr == 19)

array\_reg[19] = wdata;

else if (waddr == 20)

array\_reg[20] = wdata;

else if (waddr == 21)

array\_reg[21] = wdata;

else if (waddr == 22)

array\_reg[22] = wdata;

else if (waddr == 23)

array\_reg[23] = wdata;

else if (waddr == 24)

array\_reg[24] = wdata;

else if (waddr == 25)

array\_reg[25] = wdata;

else if (waddr == 26)

array\_reg[26] = wdata;

else if (waddr == 27)

array\_reg[27] = wdata;

else if (waddr == 28)

array\_reg[28] = wdata;

else if (waddr == 29)

array\_reg[29] = wdata;

else if (waddr == 30)

array\_reg[30] = wdata;

else if (waddr == 31)

array\_reg[31] = wdata;

else

begin

end

end

else//rf\_w==0

begin

end

end

end

always @(\*)

begin

if(raddr1==0)

rdata1=array\_reg[0];

else if(raddr1==1)

rdata1=array\_reg[1];

else if(raddr1==2)

rdata1=array\_reg[2];

else if(raddr1==3)

rdata1=array\_reg[3];

else if(raddr1==4)

rdata1=array\_reg[4];

else if(raddr1==5)

rdata1=array\_reg[5];

else if(raddr1==6)

rdata1=array\_reg[6];

else if(raddr1==7)

rdata1=array\_reg[7];

else if(raddr1==8)

rdata1=array\_reg[8];

else if(raddr1==9)

rdata1=array\_reg[9];

else if(raddr1==10)

rdata1=array\_reg[10];

else if(raddr1==11)

rdata1=array\_reg[11];

else if(raddr1==12)

rdata1=array\_reg[12];

else if(raddr1==13)

rdata1=array\_reg[13];

else if(raddr1==14)

rdata1=array\_reg[14];

else if(raddr1==15)

rdata1=array\_reg[15];

else if(raddr1==16)

rdata1=array\_reg[16];

else if(raddr1==17)

rdata1=array\_reg[17];

else if(raddr1==18)

rdata1=array\_reg[18];

else if(raddr1==19)

rdata1=array\_reg[19];

else if(raddr1==20)

rdata1=array\_reg[20];

else if(raddr1==21)

rdata1=array\_reg[21];

else if(raddr1==22)

rdata1=array\_reg[22];

else if(raddr1==23)

rdata1=array\_reg[23];

else if(raddr1==24)

rdata1=array\_reg[24];

else if(raddr1==25)

rdata1=array\_reg[25];

else if(raddr1==26)

rdata1=array\_reg[26];

else if(raddr1==27)

rdata1=array\_reg[27];

else if(raddr1==28)

rdata1=array\_reg[28];

else if(raddr1==29)

rdata1=array\_reg[29];

else if(raddr1==30)

rdata1=array\_reg[30];

else if(raddr1==31)

rdata1=array\_reg[31];

else

begin

end

if(raddr2==0)

rdata2=array\_reg[0];

else if(raddr2==1)

rdata2=array\_reg[1];

else if(raddr2==2)

rdata2=array\_reg[2];

else if(raddr2==3)

rdata2=array\_reg[3];

else if(raddr2==4)

rdata2=array\_reg[4];

else if(raddr2==5)

rdata2=array\_reg[5];

else if(raddr2==6)

rdata2=array\_reg[6];

else if(raddr2==7)

rdata2=array\_reg[7];

else if(raddr2==8)

rdata2=array\_reg[8];

else if(raddr2==9)

rdata2=array\_reg[9];

else if(raddr2==10)

rdata2=array\_reg[10];

else if(raddr2==11)

rdata2=array\_reg[11];

else if(raddr2==12)

rdata2=array\_reg[12];

else if(raddr2==13)

rdata2=array\_reg[13];

else if(raddr2==14)

rdata2=array\_reg[14];

else if(raddr2==15)

rdata2=array\_reg[15];

else if(raddr2==16)

rdata2=array\_reg[16];

else if(raddr2==17)

rdata2=array\_reg[17];

else if(raddr2==18)

rdata2=array\_reg[18];

else if(raddr2==19)

rdata2=array\_reg[19];

else if(raddr2==20)

rdata2=array\_reg[20];

else if(raddr2==21)

rdata2=array\_reg[21];

else if(raddr2==22)

rdata2=array\_reg[22];

else if(raddr2==23)

rdata2=array\_reg[23];

else if(raddr2==24)

rdata2=array\_reg[24];

else if(raddr2==25)

rdata2=array\_reg[25];

else if(raddr2==26)

rdata2=array\_reg[26];

else if(raddr2==27)

rdata2=array\_reg[27];

else if(raddr2==28)

rdata2=array\_reg[28];

else if(raddr2==29)

rdata2=array\_reg[29];

else if(raddr2==30)

rdata2=array\_reg[30];

else if(raddr2==31)

rdata2=array\_reg[31];

else

begin

end

end

endmodule

module ADD(

input [31:0]a,

input [31:0]b,

output [31:0]c

);

assign c=a+b;

endmodule

module alu(

input [31:0] a,

input [31:0] b,

input [3:0] aluc,

input reset,

output reg [31:0] r,

output reg zero,

output reg carry,

output reg negative,

output reg overflow

);

reg [31:0]kkk;

always @(\*)

begin

if(reset==1'b1)

begin

r=32'b0;

zero=0;

carry=0;

negative=0;

overflow=0;

end

else

begin

if (aluc == 4'b0000)

begin

r=a+b;

if(r==0)

zero=1;

else

zero=0;

if(r<a||r<b)

carry=1;

else

carry=0;

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b0010)

begin

r=a+b;

if(r==0)

zero=1;

else

zero=0;

if(r[31]==1)

negative=1;

else

negative=0;

if((a[31]==0&&b[31]==0&&r[31]==1)||(a[31]==1&&b[31]==1&&r[31]==0))

overflow=1;

else

overflow=0;

end

else if (aluc == 4'b0001)

begin

r=a-b;

if(r==0)

zero=1;

else

zero=0;

if(r>a)

carry=1;

else

carry=0;

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b0011)

begin

r=a-b;

if(r==0)

zero=1;

else

zero=0;

if(r[31]==1)

negative=1;

else

negative=0;

if((a[31]==0&&b[31]==1&&r[31]==1)||(a[31]==1&&b[31]==0&&r[31]==0))

overflow=1;

else

overflow=0;

end

else if (aluc == 4'b0100)

begin

r=a&b;

if(r==0)

zero=1;

else

zero=0;

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b0101)

begin

r=a|b;

if(r==0)

zero=1;

else

zero=0;

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b0110)

begin

r=a^b;

if(r==0)

zero=1;

else

zero=0;

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b0111)

begin

r=~(a|b);

if(r==0)

zero=1;

else

zero=0;

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b1000||aluc == 4'b1001)

begin

r={b[15:0],16'b0};

if(r==0)

zero=1;

else

zero=0;

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b1011)

begin

// r=(a<b)?1:0;

if(a[31]==0&&b[31]==0)

r=(a<b)?1:0;

else if(a[31]==0&&b[31]==1)

r=0;

else if(a[31]==1&&b[31]==0)

r=1;

else

r=(a<b)?1:0;

if(a-b==0)

zero=1;

else

zero=0;

kkk=a-b;

if(kkk[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b1010)

begin

r=(a<b)?1:0;

if(a-b==0)

zero=1;

else

zero=0;

if(a<b)

carry=1;

else

carry=0;

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b1100)

begin

// r=b>>>a;

r=($signed(b)) >>> a;

if(r==0)

zero=1;

else

zero=0;

if(a==0)

carry=0;

else if(a<=32)

carry=b[a-1];

else

carry=b[31];

//carry=

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b1110)

begin

r=b<<a;

if(r==0)

zero=1;

else

zero=0;

if(a==0)

carry=0;

else if(a<=32)

carry=b[32-a];

else

carry=0;

if(r[31]==1)

negative=1;

else

negative=0;

end

else if (aluc == 4'b1111)//原样传值，额外添加，用于mfhi,mflo操作中。

begin

r=a;

end

else if (aluc == 4'b1101)

begin

r=b>>a;

if(r==0)

zero=1;

else

zero=0;

if(a==0)

carry=0;

else if(a<=32)

carry=b[a-1];

else

carry=0;

//carry=

if(r[31]==1)

negative=1;

else

negative=0;

end

else

begin

end

end

end

endmodule

module controller(

/\*段间流水指令寄存器\*/

input rst,

input [31:0] if\_Instr,

input [31:0] id\_Instr,

input [31:0] ex\_Instr,

input [31:0] me\_Instr,

input [31:0] wb\_Instr,

input equal,//相等为1，不相等为0

/\*ALU状态信号，决定是否写入等操作\*/

/\*input me\_Z,

input me\_C,

input me\_N,

input me\_O,

input wb\_Z,

input wb\_C,

input wb\_N,

input wb\_O,\*/

output RF\_W,

output [3:0] ALUC,//alu识别码

/\*选择器\*/

output [1:0] PCMux,

output [2:0] ALUaMux,

output [1:0] ALUbMux,

output rdMux,

output [1:0] rdcMux,

/\*dmem信号\*/

output CS,

output DM\_R,

output DM\_W,

output reg [4:0] stall,

/\*动态流水线新增加控制信号,后两个和数据冲突相关\*/

output HI\_W,

output LO\_W,

output reg [1:0] rsoutMux,

output reg [1:0] rtoutMux,

output reg [1:0] hioutMux,

output reg [1:0] looutMux

);

/\*主体思想：分段指令译码，每一个流水段间的译码过程相同（整体替换关键字即可）\*/

//定义id分段的指令译码

wire [5:0] id\_op = id\_Instr[31:26];

wire [5:0] id\_func = id\_Instr[5:0];

//指令种类，组合逻辑构建

wire id\_add,id\_addi,id\_addu,id\_addiu,id\_sub,id\_subu;

wire id\_and,id\_andi,id\_or,id\_ori,id\_xor,id\_xori,id\_nor;

wire id\_lui;

wire id\_sll,id\_sllv,id\_sra,id\_srav,id\_srl,id\_srlv;

wire id\_slt,id\_slti,id\_sltu,id\_sltiu;

wire id\_beq,id\_bne;

wire id\_j,id\_jal,id\_jr;

wire id\_lw,id\_sw;

wire id\_mult,id\_mflo,id\_mfhi;

//R-TYPE

assign id\_add=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(~id\_func[3])&(~id\_func[2])&(~id\_func[1])&(~id\_func[0]);

assign id\_addu=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(~id\_func[3])&(~id\_func[2])&(~id\_func[1])&(id\_func[0]);

assign id\_sub=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(~id\_func[3])&(~id\_func[2])&(id\_func[1])&(~id\_func[0]);

assign id\_subu=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(~id\_func[3])&(~id\_func[2])&(id\_func[1])&(id\_func[0]);

assign id\_and=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(~id\_func[3])&(id\_func[2])&(~id\_func[1])&(~id\_func[0]);

assign id\_or=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(~id\_func[3])&(id\_func[2])&(~id\_func[1])&(id\_func[0]);

assign id\_xor=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(~id\_func[3])&(id\_func[2])&(id\_func[1])&(~id\_func[0]);

assign id\_nor=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(~id\_func[3])&(id\_func[2])&(id\_func[1])&(id\_func[0]);

assign id\_slt=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(id\_func[3])&(~id\_func[2])&(id\_func[1])&(~id\_func[0]);

assign id\_sltu=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(id\_func[5])&(~id\_func[4])&(id\_func[3])&(~id\_func[2])&(id\_func[1])&(id\_func[0]);

assign id\_sll=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(~id\_func[4])&(~id\_func[3])&(~id\_func[2])&(~id\_func[1])&(~id\_func[0]);

assign id\_srl=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(~id\_func[4])&(~id\_func[3])&(~id\_func[2])&(id\_func[1])&(~id\_func[0]);

assign id\_sra=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(~id\_func[4])&(~id\_func[3])&(~id\_func[2])&(id\_func[1])&(id\_func[0]);

assign id\_sllv=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(~id\_func[4])&(~id\_func[3])&(id\_func[2])&(~id\_func[1])&(~id\_func[0]);

assign id\_srlv=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(~id\_func[4])&(~id\_func[3])&(id\_func[2])&(id\_func[1])&(~id\_func[0]);

assign id\_srav=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(~id\_func[4])&(~id\_func[3])&(id\_func[2])&(id\_func[1])&(id\_func[0]);

assign id\_jr=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(~id\_func[4])&(id\_func[3])&(~id\_func[2])&(~id\_func[1])&(~id\_func[0]);

//I-TYPE

assign id\_addi=(~id\_op[5])&(~id\_op[4])&(id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0]);

assign id\_addiu=(~id\_op[5])&(~id\_op[4])&(id\_op[3])&(~id\_op[2])&(~id\_op[1])&(id\_op[0]);

assign id\_andi=(~id\_op[5])&(~id\_op[4])&(id\_op[3])&(id\_op[2])&(~id\_op[1])&(~id\_op[0]);

assign id\_ori=(~id\_op[5])&(~id\_op[4])&(id\_op[3])&(id\_op[2])&(~id\_op[1])&(id\_op[0]);

assign id\_xori=(~id\_op[5])&(~id\_op[4])&(id\_op[3])&(id\_op[2])&(id\_op[1])&(~id\_op[0]);

assign id\_lui=(~id\_op[5])&(~id\_op[4])&(id\_op[3])&(id\_op[2])&(id\_op[1])&(id\_op[0]);

assign id\_lw=(id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(id\_op[1])&(id\_op[0]);

assign id\_sw=(id\_op[5])&(~id\_op[4])&(id\_op[3])&(~id\_op[2])&(id\_op[1])&(id\_op[0]);

assign id\_beq=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(id\_op[2])&(~id\_op[1])&(~id\_op[0]);

assign id\_bne=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(id\_op[2])&(~id\_op[1])&(id\_op[0]);

assign id\_slti=(~id\_op[5])&(~id\_op[4])&(id\_op[3])&(~id\_op[2])&(id\_op[1])&(~id\_op[0]);

assign id\_sltiu=(~id\_op[5])&(~id\_op[4])&(id\_op[3])&(~id\_op[2])&(id\_op[1])&(id\_op[0]);

//J-TYPE

assign id\_j=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(id\_op[1])&(~id\_op[0]);

assign id\_jal=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(id\_op[1])&(id\_op[0]);

//stall-TYPE

wire id\_extra\_stall;

assign id\_extra\_stall=(id\_op[5])&(id\_op[4])&(id\_op[3])&(id\_op[2])&(id\_op[1])&(id\_op[0]);

//dynamic-extra

assign id\_mult=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(id\_func[4])&(id\_func[3])&(~id\_func[2])&(~id\_func[1])&(~id\_func[0]);

assign id\_mfhi=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(id\_func[4])&(~id\_func[3])&(~id\_func[2])&(~id\_func[1])&(~id\_func[0]);

assign id\_mflo=(~id\_op[5])&(~id\_op[4])&(~id\_op[3])&(~id\_op[2])&(~id\_op[1])&(~id\_op[0])&(~id\_func[5])&(id\_func[4])&(~id\_func[3])&(~id\_func[2])&(id\_func[1])&(~id\_func[0]);

//定义ex分段的指令译码

wire [5:0] ex\_op = ex\_Instr[31:26];

wire [5:0] ex\_func = ex\_Instr[5:0];

//指令种类，组合逻辑构建

wire ex\_add,ex\_addi,ex\_addu,ex\_addiu,ex\_sub,ex\_subu;

wire ex\_and,ex\_andi,ex\_or,ex\_ori,ex\_xor,ex\_xori,ex\_nor;

wire ex\_lui;

wire ex\_sll,ex\_sllv,ex\_sra,ex\_srav,ex\_srl,ex\_srlv;

wire ex\_slt,ex\_slti,ex\_sltu,ex\_sltiu;

wire ex\_beq,ex\_bne;

wire ex\_j,ex\_jal,ex\_jr;

wire ex\_lw,ex\_sw;

wire ex\_mult,ex\_mflo,ex\_mfhi;

//R-TYPE

assign ex\_add=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(~ex\_func[2])&(~ex\_func[1])&(~ex\_func[0]);

assign ex\_addu=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(~ex\_func[2])&(~ex\_func[1])&(ex\_func[0]);

assign ex\_sub=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(~ex\_func[2])&(ex\_func[1])&(~ex\_func[0]);

assign ex\_subu=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(~ex\_func[2])&(ex\_func[1])&(ex\_func[0]);

assign ex\_and=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(ex\_func[2])&(~ex\_func[1])&(~ex\_func[0]);

assign ex\_or=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(ex\_func[2])&(~ex\_func[1])&(ex\_func[0]);

assign ex\_xor=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(ex\_func[2])&(ex\_func[1])&(~ex\_func[0]);

assign ex\_nor=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(ex\_func[2])&(ex\_func[1])&(ex\_func[0]);

assign ex\_slt=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(ex\_func[3])&(~ex\_func[2])&(ex\_func[1])&(~ex\_func[0]);

assign ex\_sltu=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(ex\_func[5])&(~ex\_func[4])&(ex\_func[3])&(~ex\_func[2])&(ex\_func[1])&(ex\_func[0]);

assign ex\_sll=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(~ex\_func[2])&(~ex\_func[1])&(~ex\_func[0]);

assign ex\_srl=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(~ex\_func[2])&(ex\_func[1])&(~ex\_func[0]);

assign ex\_sra=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(~ex\_func[2])&(ex\_func[1])&(ex\_func[0]);

assign ex\_sllv=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(ex\_func[2])&(~ex\_func[1])&(~ex\_func[0]);

assign ex\_srlv=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(ex\_func[2])&(ex\_func[1])&(~ex\_func[0]);

assign ex\_srav=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(~ex\_func[4])&(~ex\_func[3])&(ex\_func[2])&(ex\_func[1])&(ex\_func[0]);

assign ex\_jr=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(~ex\_func[4])&(ex\_func[3])&(~ex\_func[2])&(~ex\_func[1])&(~ex\_func[0]);

//I-TYPE

assign ex\_addi=(~ex\_op[5])&(~ex\_op[4])&(ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0]);

assign ex\_addiu=(~ex\_op[5])&(~ex\_op[4])&(ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(ex\_op[0]);

assign ex\_andi=(~ex\_op[5])&(~ex\_op[4])&(ex\_op[3])&(ex\_op[2])&(~ex\_op[1])&(~ex\_op[0]);

assign ex\_ori=(~ex\_op[5])&(~ex\_op[4])&(ex\_op[3])&(ex\_op[2])&(~ex\_op[1])&(ex\_op[0]);

assign ex\_xori=(~ex\_op[5])&(~ex\_op[4])&(ex\_op[3])&(ex\_op[2])&(ex\_op[1])&(~ex\_op[0]);

assign ex\_lui=(~ex\_op[5])&(~ex\_op[4])&(ex\_op[3])&(ex\_op[2])&(ex\_op[1])&(ex\_op[0]);

assign ex\_lw=(ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(ex\_op[1])&(ex\_op[0]);

assign ex\_sw=(ex\_op[5])&(~ex\_op[4])&(ex\_op[3])&(~ex\_op[2])&(ex\_op[1])&(ex\_op[0]);

assign ex\_beq=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(ex\_op[2])&(~ex\_op[1])&(~ex\_op[0]);

assign ex\_bne=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(ex\_op[2])&(~ex\_op[1])&(ex\_op[0]);

assign ex\_slti=(~ex\_op[5])&(~ex\_op[4])&(ex\_op[3])&(~ex\_op[2])&(ex\_op[1])&(~ex\_op[0]);

assign ex\_sltiu=(~ex\_op[5])&(~ex\_op[4])&(ex\_op[3])&(~ex\_op[2])&(ex\_op[1])&(ex\_op[0]);

//J-TYPE

assign ex\_j=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(ex\_op[1])&(~ex\_op[0]);

assign ex\_jal=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(ex\_op[1])&(ex\_op[0]);

//stall-TYPE

wire ex\_extra\_stall;

assign ex\_extra\_stall=(ex\_op[5])&(ex\_op[4])&(ex\_op[3])&(ex\_op[2])&(ex\_op[1])&(ex\_op[0]);

//dynamic-extra

assign ex\_mult=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(ex\_func[4])&(ex\_func[3])&(~ex\_func[2])&(~ex\_func[1])&(~ex\_func[0]);

assign ex\_mfhi=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(ex\_func[4])&(~ex\_func[3])&(~ex\_func[2])&(~ex\_func[1])&(~ex\_func[0]);

assign ex\_mflo=(~ex\_op[5])&(~ex\_op[4])&(~ex\_op[3])&(~ex\_op[2])&(~ex\_op[1])&(~ex\_op[0])&(~ex\_func[5])&(ex\_func[4])&(~ex\_func[3])&(~ex\_func[2])&(ex\_func[1])&(~ex\_func[0]);

//定义me分段的指令译码

wire [5:0] me\_op = me\_Instr[31:26];

wire [5:0] me\_func = me\_Instr[5:0];

//指令种类，组合逻辑构建

wire me\_add,me\_addi,me\_addu,me\_addiu,me\_sub,me\_subu;

wire me\_and,me\_andi,me\_or,me\_ori,me\_xor,me\_xori,me\_nor;

wire me\_lui;

wire me\_sll,me\_sllv,me\_sra,me\_srav,me\_srl,me\_srlv;

wire me\_slt,me\_slti,me\_sltu,me\_sltiu;

wire me\_beq,me\_bne;

wire me\_j,me\_jal,me\_jr;

wire me\_lw,me\_sw;

wire me\_mult,me\_mflo,me\_mfhi;

//R-TYPE

assign me\_add=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(~me\_func[3])&(~me\_func[2])&(~me\_func[1])&(~me\_func[0]);

assign me\_addu=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(~me\_func[3])&(~me\_func[2])&(~me\_func[1])&(me\_func[0]);

assign me\_sub=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(~me\_func[3])&(~me\_func[2])&(me\_func[1])&(~me\_func[0]);

assign me\_subu=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(~me\_func[3])&(~me\_func[2])&(me\_func[1])&(me\_func[0]);

assign me\_and=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(~me\_func[3])&(me\_func[2])&(~me\_func[1])&(~me\_func[0]);

assign me\_or=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(~me\_func[3])&(me\_func[2])&(~me\_func[1])&(me\_func[0]);

assign me\_xor=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(~me\_func[3])&(me\_func[2])&(me\_func[1])&(~me\_func[0]);

assign me\_nor=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(~me\_func[3])&(me\_func[2])&(me\_func[1])&(me\_func[0]);

assign me\_slt=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(me\_func[3])&(~me\_func[2])&(me\_func[1])&(~me\_func[0]);

assign me\_sltu=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(me\_func[5])&(~me\_func[4])&(me\_func[3])&(~me\_func[2])&(me\_func[1])&(me\_func[0]);

assign me\_sll=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(~me\_func[4])&(~me\_func[3])&(~me\_func[2])&(~me\_func[1])&(~me\_func[0]);

assign me\_srl=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(~me\_func[4])&(~me\_func[3])&(~me\_func[2])&(me\_func[1])&(~me\_func[0]);

assign me\_sra=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(~me\_func[4])&(~me\_func[3])&(~me\_func[2])&(me\_func[1])&(me\_func[0]);

assign me\_sllv=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(~me\_func[4])&(~me\_func[3])&(me\_func[2])&(~me\_func[1])&(~me\_func[0]);

assign me\_srlv=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(~me\_func[4])&(~me\_func[3])&(me\_func[2])&(me\_func[1])&(~me\_func[0]);

assign me\_srav=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(~me\_func[4])&(~me\_func[3])&(me\_func[2])&(me\_func[1])&(me\_func[0]);

assign me\_jr=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(~me\_func[4])&(me\_func[3])&(~me\_func[2])&(~me\_func[1])&(~me\_func[0]);

//I-TYPE

assign me\_addi=(~me\_op[5])&(~me\_op[4])&(me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0]);

assign me\_addiu=(~me\_op[5])&(~me\_op[4])&(me\_op[3])&(~me\_op[2])&(~me\_op[1])&(me\_op[0]);

assign me\_andi=(~me\_op[5])&(~me\_op[4])&(me\_op[3])&(me\_op[2])&(~me\_op[1])&(~me\_op[0]);

assign me\_ori=(~me\_op[5])&(~me\_op[4])&(me\_op[3])&(me\_op[2])&(~me\_op[1])&(me\_op[0]);

assign me\_xori=(~me\_op[5])&(~me\_op[4])&(me\_op[3])&(me\_op[2])&(me\_op[1])&(~me\_op[0]);

assign me\_lui=(~me\_op[5])&(~me\_op[4])&(me\_op[3])&(me\_op[2])&(me\_op[1])&(me\_op[0]);

assign me\_lw=(me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(me\_op[1])&(me\_op[0]);

assign me\_sw=(me\_op[5])&(~me\_op[4])&(me\_op[3])&(~me\_op[2])&(me\_op[1])&(me\_op[0]);

assign me\_beq=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(me\_op[2])&(~me\_op[1])&(~me\_op[0]);

assign me\_bne=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(me\_op[2])&(~me\_op[1])&(me\_op[0]);

assign me\_slti=(~me\_op[5])&(~me\_op[4])&(me\_op[3])&(~me\_op[2])&(me\_op[1])&(~me\_op[0]);

assign me\_sltiu=(~me\_op[5])&(~me\_op[4])&(me\_op[3])&(~me\_op[2])&(me\_op[1])&(me\_op[0]);

//J-TYPE

assign me\_j=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(me\_op[1])&(~me\_op[0]);

assign me\_jal=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(me\_op[1])&(me\_op[0]);

//stall-TYPE

wire me\_extra\_stall;

assign me\_extra\_stall=(me\_op[5])&(me\_op[4])&(me\_op[3])&(me\_op[2])&(me\_op[1])&(me\_op[0]);

//dynamic-extra

assign me\_mult=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(me\_func[4])&(me\_func[3])&(~me\_func[2])&(~me\_func[1])&(~me\_func[0]);

assign me\_mfhi=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(me\_func[4])&(~me\_func[3])&(~me\_func[2])&(~me\_func[1])&(~me\_func[0]);

assign me\_mflo=(~me\_op[5])&(~me\_op[4])&(~me\_op[3])&(~me\_op[2])&(~me\_op[1])&(~me\_op[0])&(~me\_func[5])&(me\_func[4])&(~me\_func[3])&(~me\_func[2])&(me\_func[1])&(~me\_func[0]);

//定义ex分段的指令译码

wire [5:0] wb\_op = wb\_Instr[31:26];

wire [5:0] wb\_func = wb\_Instr[5:0];

//指令种类，组合逻辑构建

wire wb\_add,wb\_addi,wb\_addu,wb\_addiu,wb\_sub,wb\_subu;

wire wb\_and,wb\_andi,wb\_or,wb\_ori,wb\_xor,wb\_xori,wb\_nor;

wire wb\_lui;

wire wb\_sll,wb\_sllv,wb\_sra,wb\_srav,wb\_srl,wb\_srlv;

wire wb\_slt,wb\_slti,wb\_sltu,wb\_sltiu;

wire wb\_beq,wb\_bne;

wire wb\_j,wb\_jal,wb\_jr;

wire wb\_lw,wb\_sw;

wire wb\_mult,wb\_mflo,wb\_mfhi;

//R-TYPE

assign wb\_add=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(~wb\_func[2])&(~wb\_func[1])&(~wb\_func[0]);

assign wb\_addu=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(~wb\_func[2])&(~wb\_func[1])&(wb\_func[0]);

assign wb\_sub=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(~wb\_func[2])&(wb\_func[1])&(~wb\_func[0]);

assign wb\_subu=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(~wb\_func[2])&(wb\_func[1])&(wb\_func[0]);

assign wb\_and=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(wb\_func[2])&(~wb\_func[1])&(~wb\_func[0]);

assign wb\_or=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(wb\_func[2])&(~wb\_func[1])&(wb\_func[0]);

assign wb\_xor=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(wb\_func[2])&(wb\_func[1])&(~wb\_func[0]);

assign wb\_nor=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(wb\_func[2])&(wb\_func[1])&(wb\_func[0]);

assign wb\_slt=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(wb\_func[3])&(~wb\_func[2])&(wb\_func[1])&(~wb\_func[0]);

assign wb\_sltu=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(wb\_func[5])&(~wb\_func[4])&(wb\_func[3])&(~wb\_func[2])&(wb\_func[1])&(wb\_func[0]);

assign wb\_sll=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(~wb\_func[2])&(~wb\_func[1])&(~wb\_func[0]);

assign wb\_srl=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(~wb\_func[2])&(wb\_func[1])&(~wb\_func[0]);

assign wb\_sra=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(~wb\_func[2])&(wb\_func[1])&(wb\_func[0]);

assign wb\_sllv=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(wb\_func[2])&(~wb\_func[1])&(~wb\_func[0]);

assign wb\_srlv=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(wb\_func[2])&(wb\_func[1])&(~wb\_func[0]);

assign wb\_srav=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(~wb\_func[4])&(~wb\_func[3])&(wb\_func[2])&(wb\_func[1])&(wb\_func[0]);

assign wb\_jr=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(~wb\_func[4])&(wb\_func[3])&(~wb\_func[2])&(~wb\_func[1])&(~wb\_func[0]);

//I-TYPE

assign wb\_addi=(~wb\_op[5])&(~wb\_op[4])&(wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0]);

assign wb\_addiu=(~wb\_op[5])&(~wb\_op[4])&(wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(wb\_op[0]);

assign wb\_andi=(~wb\_op[5])&(~wb\_op[4])&(wb\_op[3])&(wb\_op[2])&(~wb\_op[1])&(~wb\_op[0]);

assign wb\_ori=(~wb\_op[5])&(~wb\_op[4])&(wb\_op[3])&(wb\_op[2])&(~wb\_op[1])&(wb\_op[0]);

assign wb\_xori=(~wb\_op[5])&(~wb\_op[4])&(wb\_op[3])&(wb\_op[2])&(wb\_op[1])&(~wb\_op[0]);

assign wb\_lui=(~wb\_op[5])&(~wb\_op[4])&(wb\_op[3])&(wb\_op[2])&(wb\_op[1])&(wb\_op[0]);

assign wb\_lw=(wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(wb\_op[1])&(wb\_op[0]);

assign wb\_sw=(wb\_op[5])&(~wb\_op[4])&(wb\_op[3])&(~wb\_op[2])&(wb\_op[1])&(wb\_op[0]);

assign wb\_beq=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(wb\_op[2])&(~wb\_op[1])&(~wb\_op[0]);

assign wb\_bne=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(wb\_op[2])&(~wb\_op[1])&(wb\_op[0]);

assign wb\_slti=(~wb\_op[5])&(~wb\_op[4])&(wb\_op[3])&(~wb\_op[2])&(wb\_op[1])&(~wb\_op[0]);

assign wb\_sltiu=(~wb\_op[5])&(~wb\_op[4])&(wb\_op[3])&(~wb\_op[2])&(wb\_op[1])&(wb\_op[0]);

//J-TYPE

assign wb\_j=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(wb\_op[1])&(~wb\_op[0]);

assign wb\_jal=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(wb\_op[1])&(wb\_op[0]);

//stall-TYPE

wire wb\_extra\_stall;

assign wb\_extra\_stall=(wb\_op[5])&(wb\_op[4])&(wb\_op[3])&(wb\_op[2])&(wb\_op[1])&(wb\_op[0]);

//dynamic-extra

assign wb\_mult=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(wb\_func[4])&(wb\_func[3])&(~wb\_func[2])&(~wb\_func[1])&(~wb\_func[0]);

assign wb\_mfhi=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(wb\_func[4])&(~wb\_func[3])&(~wb\_func[2])&(~wb\_func[1])&(~wb\_func[0]);

assign wb\_mflo=(~wb\_op[5])&(~wb\_op[4])&(~wb\_op[3])&(~wb\_op[2])&(~wb\_op[1])&(~wb\_op[0])&(~wb\_func[5])&(wb\_func[4])&(~wb\_func[3])&(~wb\_func[2])&(wb\_func[1])&(~wb\_func[0]);

/\*在初始的id段给出所有的控制信号，在后续过程中，使用段间流水寄存器传递段间的控制值\*/

/\*本来为：((~O)&(i\_add|i\_addi|i\_sub))，在后续流水线到达WB阶段时再处理\*/

assign RF\_W=((id\_add|id\_addi|id\_sub))|(id\_addu)|(id\_addiu)|(id\_subu)|(id\_and)|(id\_andi)|(id\_or)|(id\_ori)|

(id\_xor)|(id\_xori)|(id\_nor)|(id\_lui)|(id\_sll)|(id\_sllv)|(id\_sra)|(id\_srav)|(id\_srl)|(id\_srlv)|

(id\_slt)|(id\_slti)|(id\_sltu)|(id\_sltiu)|(id\_jal)|(id\_lw)|(id\_mflo)|(id\_mfhi);

assign ALUC[3]=(id\_lui)|(id\_sll)|(id\_sllv)|(id\_sra)|(id\_srav)|(id\_srl)|(id\_srlv)|(id\_slt)|(id\_slti)|(id\_sltu)|(id\_sltiu)|(id\_mflo)|(id\_mfhi);

assign ALUC[2]=(id\_and)|(id\_andi)|(id\_or)|(id\_ori)|(id\_xor)|(id\_xori)|(id\_nor)|(id\_sll)|(id\_sllv)|(id\_sra)|(id\_srav)|(id\_srl)|(id\_srlv)|(id\_mflo)|(id\_mfhi);

assign ALUC[1]=(id\_add)|(id\_addi)|(id\_sub)|(id\_xor)|(id\_xori)|(id\_nor)|(id\_sll)|(id\_sllv)|(id\_slt)|(id\_slti)|(id\_sltu)|(id\_sltiu)|(id\_lw)|(id\_sw)|(id\_mflo)|(id\_mfhi);

assign ALUC[0]=(id\_sub)|(id\_subu)|(id\_or)|(id\_ori)|(id\_nor)|(id\_srl)|(id\_srlv)|(id\_slt)|(id\_slti)|(id\_mflo)|(id\_mfhi);

assign PCMux[1]=id\_j|id\_jal|id\_jr;

assign PCMux[0]=(equal&id\_beq)|((~equal)&id\_bne)|id\_jr;

//hi:011 lo:100

assign ALUaMux[2]=id\_mflo;

assign ALUaMux[1]=id\_jal|id\_mfhi;//npc:10

assign ALUaMux[0]=(id\_add)|(id\_addu)|(id\_addi)|(id\_addiu)|(id\_sub)|(id\_subu)|(id\_and)|(id\_andi)|(id\_or)|(id\_ori)|

(id\_xor)|(id\_xori)|(id\_nor)|(id\_lui)|(id\_sllv)|(id\_srav)|(id\_srlv)|

(id\_slt)|(id\_slti)|(id\_sltu)|(id\_sltiu)|(id\_beq)|(id\_bne)|(id\_j)|(id\_jr)|(id\_lw)|(id\_sw)|id\_mfhi|id\_mult;

assign ALUbMux[1]=(id\_addi)|(id\_addiu)|(id\_slti)|(id\_sltiu)|(id\_lw)|(id\_sw)|id\_jal;

assign ALUbMux[0]=(id\_andi)|(id\_ori)|(id\_xori)|(id\_lui)|id\_jal;

//id\_jal:11,alub=0

assign rdMux=id\_lw;

assign rdcMux[1]=id\_jal;

assign rdcMux[0]=(id\_addi)|(id\_addiu)|(id\_andi)|(id\_ori)|

(id\_xori)|(id\_lui)|

(id\_slti)|(id\_sltiu)|(id\_beq)|(id\_bne)|(id\_j)|(id\_jr)|(id\_lw)|(id\_sw);

assign CS=(id\_lw)|(id\_sw);

assign DM\_R=(id\_lw);

assign DM\_W=(id\_sw);

assign HI\_W=id\_mfhi;

assign LO\_W=id\_mflo;

/\*控制信号添加完毕，接下来检测流水线读写数据相关性

由于是静态流水线，因此只有先写后读相关

在wb周期向寄存器写入，同一周期，半段写，半段读，先写后读，所以不用考虑wb和id之间关系

内存中，读写在同一个周期，没有问题

只需要考虑id中读和ex，me，wb中写，两个周期内会有冲突\*/

/\*分层次：mult，mfhi，mflo一个层次\*/

/\*两个解决方案，在alu中能够计算出来的就直接算，传值。

跳转或内存读取指令则暂停。\*/

wire id\_read\_rs;//id段是否读rs

wire id\_read\_rt;//id段是否读rt

reg ex\_write\_rd;//ex段是否写rd

reg [4:0] ex\_write\_rdc;//（如果是），ex段写rd的地址

reg me\_write\_rd;

reg [4:0] me\_write\_rdc;

reg wb\_write\_rd;

reg [4:0] wb\_write\_rdc;

reg ex\_if\_jal\_or\_lw;//是否jal或lw

reg me\_if\_jal\_or\_lw;//是否jal或lw

reg wb\_if\_jal\_or\_lw;//是否jal或lw

/\*

(id\_add|id\_addi|id\_sub)|(id\_addu)|(id\_addiu)|(id\_subu)|(id\_and)|(id\_andi)|(id\_or)|(id\_ori)|

(id\_xor)|(id\_xori)|(id\_nor)|(id\_lui)|(id\_sll)|(id\_sllv)|(id\_sra)|(id\_srav)|(id\_srl)|(id\_srlv)|

(id\_slt)|(id\_slti)|(id\_sltu)|(id\_sltiu)|(id\_jal)|(id\_lw)|(id\_beq)|(id\_bne)|(id\_j)|(id\_jr)|(id\_sw);

\*/

assign id\_read\_rs=(id\_add|id\_addi|id\_sub)|(id\_addu)|(id\_addiu)|(id\_subu)|(id\_and)|(id\_andi)|(id\_or)|(id\_ori)|

(id\_xor)|(id\_xori)|(id\_nor)|(id\_sllv)|(id\_srav)|(id\_srlv)|

(id\_slt)|(id\_slti)|(id\_sltu)|(id\_sltiu)|(id\_lw)|(id\_beq)|(id\_bne)|(id\_jr)|(id\_sw)|(id\_mult);

assign id\_read\_rt=(id\_add|id\_sub)|(id\_addu)|(id\_subu)|(id\_and)|(id\_or)|

(id\_xor)|(id\_nor)|(id\_sll)|(id\_sllv)|(id\_sra)|(id\_srav)|(id\_srl)|(id\_srlv)|

(id\_slt)|(id\_sltu)|(id\_beq)|(id\_bne)|(id\_sw)|(id\_mult);

/\*assign ex\_write\_rd=(ex\_add|ex\_sub)|(ex\_addu)|(ex\_subu)|(ex\_and)|(ex\_or)|

(ex\_xor)|(ex\_nor)|(ex\_sll)|(ex\_sllv)|(ex\_sra)|(ex\_srav)|(ex\_srl)|(ex\_srlv)|

(ex\_slt)|(ex\_sltu);\*/

wire [4:0] id\_rs = id\_Instr[25:21];

wire [4:0] id\_rt = id\_Instr[20:16];

wire [4:0] id\_rd = id\_Instr[15:11];

wire [4:0] ex\_rs = ex\_Instr[25:21];

wire [4:0] ex\_rt = ex\_Instr[20:16];

wire [4:0] ex\_rd = ex\_Instr[15:11];

wire [4:0] me\_rs = me\_Instr[25:21];

wire [4:0] me\_rt = me\_Instr[20:16];

wire [4:0] me\_rd = me\_Instr[15:11];

wire [4:0] wb\_rs = wb\_Instr[25:21];

wire [4:0] wb\_rt = wb\_Instr[20:16];

wire [4:0] wb\_rd = wb\_Instr[15:11];

always @ (\*)

begin

if((ex\_add|ex\_sub)|(ex\_addu)|(ex\_subu)|(ex\_and)|(ex\_or)|(ex\_xor)|(ex\_nor)

|(ex\_sll)|(ex\_sllv)|(ex\_sra)|(ex\_srav)|(ex\_srl)|(ex\_srlv)|(ex\_slt)|(ex\_sltu)|(ex\_mflo)|(ex\_mfhi))

begin

ex\_write\_rd=1;

ex\_write\_rdc=ex\_rd;

ex\_if\_jal\_or\_lw=0;

end

else if((ex\_addi)|(ex\_addiu)|(ex\_andi)|(ex\_ori)|(ex\_xori)|(ex\_lui)|(ex\_slti)|(ex\_sltiu)|(ex\_lw))

begin

ex\_write\_rd=1;

ex\_write\_rdc=ex\_rt;

if(ex\_lw)

ex\_if\_jal\_or\_lw=1;

else

ex\_if\_jal\_or\_lw=0;

end

else if(ex\_jal)

begin

ex\_write\_rd=1;

ex\_write\_rdc=5'b11111;

ex\_if\_jal\_or\_lw=1;

end

else

begin

ex\_write\_rd=0;

ex\_write\_rdc=5'b00000;

ex\_if\_jal\_or\_lw=0;

end

end

always @ (\*)

begin

if((me\_add|me\_sub)|(me\_addu)|(me\_subu)|(me\_and)|(me\_or)|(me\_xor)|(me\_nor)

|(me\_sll)|(me\_sllv)|(me\_sra)|(me\_srav)|(me\_srl)|(me\_srlv)|(me\_slt)|(me\_sltu)|(me\_mflo)|(me\_mfhi))

begin

me\_write\_rd=1;

me\_write\_rdc=me\_rd;

me\_if\_jal\_or\_lw=0;

end

else if((me\_addi)|(me\_addiu)|(me\_andi)|(me\_ori)|(me\_xori)|(me\_lui)|(me\_slti)|(me\_sltiu)|(me\_lw))

begin

me\_write\_rd=1;

me\_write\_rdc=me\_rt;

if(me\_lw)

me\_if\_jal\_or\_lw=1;

else

me\_if\_jal\_or\_lw=0;

end

else if(me\_jal)

begin

me\_write\_rd=1;

me\_write\_rdc=5'b11111;

me\_if\_jal\_or\_lw=1;

end

else

begin

me\_write\_rd=0;

me\_write\_rdc=5'b00000;

me\_if\_jal\_or\_lw=0;

end

end

always @ (\*)

begin

if((wb\_add|wb\_sub)|(wb\_addu)|(wb\_subu)|(wb\_and)|(wb\_or)|(wb\_xor)|(wb\_nor)

|(wb\_sll)|(wb\_sllv)|(wb\_sra)|(wb\_srav)|(wb\_srl)|(wb\_srlv)|(wb\_slt)|(wb\_sltu)|(wb\_mflo)|(wb\_mfhi))

begin

wb\_write\_rd=1;

wb\_write\_rdc=wb\_rd;

wb\_if\_jal\_or\_lw=0;

end

else if((wb\_addi)|(wb\_addiu)|(wb\_andi)|(wb\_ori)|(wb\_xori)|(wb\_lui)|(wb\_slti)|(wb\_sltiu)|(wb\_lw))

begin

wb\_write\_rd=1;

wb\_write\_rdc=wb\_rt;

if(wb\_lw)

wb\_if\_jal\_or\_lw=1;

else

wb\_if\_jal\_or\_lw=0;

end

else if(wb\_jal)

begin

wb\_write\_rd=1;

wb\_write\_rdc=5'b11111;

wb\_if\_jal\_or\_lw=1;

end

else

begin

wb\_write\_rd=0;

wb\_write\_rdc=5'b00000;

wb\_if\_jal\_or\_lw=0;

end

end

/\* wire id\_read\_rs;//id段是否读rs

wire id\_read\_rt;//id段是否读rt

reg ex\_write\_rd;//ex段是否写rd

reg [4:0] ex\_write\_rdc;//（如果是），ex段写rd的地址

reg me\_write\_rd;

reg [4:0] me\_write\_rdc;\*/

/\*只对id-ex段的数据冲突进行处理，其他段照常暂停\*/

/\*读写相关检测完成，开始判断流水线是否暂停，在哪里暂停\*/

/\*中断机制：有多个写和最后一个读，以最后一个写为标准

因此检测顺序为：wb-me-ex,ex为最靠近当前指令的地方\*/

always @ (\*)

begin

if(rst==1)

stall=5'b00000;

stall=5'b00000;

rsoutMux=2'b00;

rtoutMux=2'b00;

if(wb\_write\_rd)

begin

if(id\_read\_rs==1 && wb\_write\_rdc==id\_rs && wb\_if\_jal\_or\_lw==0)

rsoutMux=2'b11;

if(id\_read\_rt==1 && wb\_write\_rdc==id\_rt && wb\_if\_jal\_or\_lw==0)

rtoutMux=2'b11;

if(((id\_read\_rs==1 && wb\_write\_rdc==id\_rs) || (id\_read\_rt==1 && wb\_write\_rdc==id\_rt)) && wb\_if\_jal\_or\_lw==1)

stall=5'b00011;

end

if(me\_write\_rd)

begin

if(id\_read\_rs==1 && me\_write\_rdc==id\_rs && me\_if\_jal\_or\_lw==0)

rsoutMux=2'b10;

if(id\_read\_rt==1 && me\_write\_rdc==id\_rt && me\_if\_jal\_or\_lw==0)

rtoutMux=2'b10;

if(((id\_read\_rs==1 && me\_write\_rdc==id\_rs) || (id\_read\_rt==1 && me\_write\_rdc==id\_rt)) && me\_if\_jal\_or\_lw==1)

stall=5'b00011;

end

if(ex\_write\_rd)

begin

if(id\_read\_rs==1 && ex\_write\_rdc==id\_rs && ex\_if\_jal\_or\_lw==0)

rsoutMux=2'b01;

if(id\_read\_rt==1 && ex\_write\_rdc==id\_rt && ex\_if\_jal\_or\_lw==0)

rtoutMux=2'b01;

if(((id\_read\_rs==1 && ex\_write\_rdc==id\_rs) || (id\_read\_rt==1 && ex\_write\_rdc==id\_rt)) && ex\_if\_jal\_or\_lw==1)

stall=5'b00011;

end

end

wire id\_read\_hi;//id段是否读hi

wire id\_read\_lo;//id段是否读lo

reg ex\_write\_hi;//ex段是否写hi

reg ex\_write\_lo;//ex段是否写lo

reg me\_write\_hi;//me段是否写hi

reg me\_write\_lo;//me段是否写lo

reg wb\_write\_hi;//wb段是否写hi

reg wb\_write\_lo;//wb段是否写lo

assign id\_read\_hi=(id\_mfhi);

assign id\_read\_lo=(id\_mflo);

always @ (\*)

begin

if(ex\_mult)

begin

ex\_write\_hi=1;

ex\_write\_lo=1;

end

else

begin

ex\_write\_hi=0;

ex\_write\_lo=0;

end

if(me\_mult)

begin

me\_write\_hi=1;

me\_write\_lo=1;

end

else

begin

me\_write\_hi=0;

me\_write\_lo=0;

end

if(wb\_mult)

begin

wb\_write\_hi=1;

wb\_write\_lo=1;

end

else

begin

wb\_write\_hi=0;

wb\_write\_lo=0;

end

end

always @(\*)

begin

hioutMux=2'b00;

looutMux=2'b00;

if(id\_read\_hi && wb\_write\_hi)

hioutMux=2'b11;

if(id\_read\_hi && me\_write\_hi)

hioutMux=2'b10;

if(id\_read\_hi && ex\_write\_hi)

hioutMux=2'b01;

if(id\_read\_lo && wb\_write\_lo)

looutMux=2'b11;

if(id\_read\_lo && me\_write\_lo)

looutMux=2'b10;

if(id\_read\_lo && ex\_write\_lo)

looutMux=2'b01;

end

endmodule

module cpu(

input clk,

input rst,

input [31:0] imem\_out,

input [31:0] dmem\_out,

output [31:0] imem\_in,

output [31:0] dmem\_data\_in,

output [31:0] dmem\_addr\_in,

output dmem\_DM\_W,

output [31:0] pc\_out,

output wire [31:0] add\_out,

output wire equal\_out,

output wire [31:0] join\_out,

output wire [31:0] pcmux\_out,

output wire [31:0] aluamux\_out,

output wire [31:0] alubmux\_out,

output wire [31:0] rdmux\_out,

output wire [4:0] rdcmux\_out,

output wire [31:0] ext5\_out,

output wire [31:0] ext16\_out,

output wire [31:0] sext16\_out,

output wire [31:0] sext18\_out,

output wire [31:0] rs\_out,

output wire [31:0] rt\_out,

output wire [31:0] alu\_out,

/\*控制信号\*/

/\*---------------------------------\*/

output wire RF\_W\_control,

output wire [3:0] ALUC\_control,//alu识别码

/\*选择器\*/

output wire [1:0] PCMux\_control,

output wire [2:0] ALUaMux\_control,

output wire [1:0] ALUbMux\_control,

output wire rdMux\_control,

output wire [1:0] rdcMux\_control,

/\*dmem信号\*/

output wire CS\_control,//无用

output wire DM\_R\_control,//无用

output wire DM\_W\_control,

output wire [4:0] stall,

/\*动态额外添加\*/

output wire HI\_W,

output wire LO\_W,

output wire [1:0] rsoutMux,

output wire [1:0] rtoutMux,

output wire [1:0] hioutMux,

output wire [1:0] looutMux,

output wire [31:0] rs\_to\_mux,

output wire [31:0] rt\_to\_mux,

output wire [31:0] hi\_out,

output wire [31:0] lo\_out,

output wire [31:0] hi\_mux\_out,

output wire [31:0] lo\_mux\_out,

output wire [63:0] z,

output reg [63:0] Z1,

output reg [63:0] Z2,

/\*id-ex段间控制信号寄存器\*/

output reg id\_ex\_RF\_W,

output reg [3:0] id\_ex\_aluc,

output reg id\_ex\_DM\_W,

output reg id\_ex\_rdMux,

output reg [1:0] id\_ex\_rdcMux,

/\*ex-me段间控制信号寄存器\*/

output reg ex\_me\_RF\_W,

// output reg [3:0] ex\_me\_aluc,

output reg ex\_me\_DM\_W,

output reg ex\_me\_rdMux,

output reg [1:0] ex\_me\_rdcMux,

output wire ex\_me\_Z,

output wire ex\_me\_C,

output wire ex\_me\_N,

output wire ex\_me\_O,

/\*me-wb段间控制信号寄存器\*/

output reg me\_wb\_RF\_W,

// output reg [3:0] me\_wb\_aluc,

//output reg me\_wb\_DM\_W,

output reg me\_wb\_rdMux,

output reg [1:0] me\_wb\_rdcMux,

output reg me\_wb\_Z,

output reg me\_wb\_C,

output reg me\_wb\_N,

output reg me\_wb\_O,

/\*---------------------------------\*/

/\*段间寄存器\*/

output reg [31:0] NPC,

output reg [31:0] IR1,

output reg [31:0] IR2,

output reg [31:0] IR3,

output reg [31:0] IR4,

output reg [31:0] ALUa,

output reg [31:0] ALUb,

output reg [31:0] ALUo1,

output reg [31:0] ALUo2,

output reg [31:0] Rdata1,

output reg [31:0] Rdata2,

output reg [31:0] Wdata,

output reg [31:0] pc1,

output reg [31:0] pc2,

output reg [31:0] pc3,

output reg [31:0] pc4,

output [31:0] out\_seg

);

/\*流水线：分段声明；分段定义、实例化模块\*/

/\*每个流水段间，处理两个东西：

1：段间寄存器

2：之后所有流水段所需要的控制信号\*/

/\*控制信号：id段译码得出，在id-ex，ex-me，me-wb三个段间需要使用\*/

//wire [31:0] pc\_out;

//wire [31:0] npc\_out;

//wire [31:0] imem\_out;

//wire [31:0] dmem\_out;

wire [31:0] outreg1;

wire [31:0] outreg2;

assign out\_seg=outreg1;

/\*组合逻辑控制器\*/

controller con(rst,imem\_out,IR1,IR2,IR3,IR4,equal\_out,RF\_W\_control,ALUC\_control,

PCMux\_control,ALUaMux\_control,ALUbMux\_control,rdMux\_control,rdcMux\_control,

CS\_control,DM\_R\_control,DM\_W\_control,stall,

HI\_W,LO\_W,rsoutMux,rtoutMux,hioutMux,looutMux);

//stall:[4:0] wb me ex id if

//stall后方正常流水，前方停止

//if

selector41 PCmux(NPC,add\_out,join\_out,rs\_out,PCMux\_control,pcmux\_out);

pc pc\_cpu(clk,rst,~(stall[0]),pcmux\_out,pc\_out);

assign imem\_in=pc\_out;

//if-id

always @ (posedge clk or posedge rst) begin

if (rst==1 || (stall[0]==1 && stall[1]==0)) begin

NPC <= 32'h00400000;

IR1 <= 32'hffffffff;

end

else if ((stall[0]==0)) begin

NPC <= pc\_out + 4;

IR1 <= imem\_out;

pc1<=pc\_out;

end

else

begin

/\*NPC <= pc\_out + 4;

IR1 <= imem\_out;\*/

end

end

//id

Regfiles cpu\_ref(clk,rst,me\_wb\_RF\_W,IR1[25:21],IR1[20:16],rdcmux\_out,rdmux\_out,rs\_to\_mux,rt\_to\_mux,outreg1,outreg2);

equal\_unit eu(rs\_out,rt\_out,equal\_out);

joint\_pc\_im jpi({IR1[25:0],2'b00},pc\_out[31:28],join\_out);

ADD add\_unit(sext18\_out,NPC,add\_out);

EXT\_5 ext5(IR1[10:6],ext5\_out);

EXT\_16 ext16(IR1[15:0],ext16\_out);

S\_EXT\_16 sext16(IR1[15:0],sext16\_out);

S\_EXT\_18 sext18({IR1[15:0],2'b00},sext18\_out);

selector81 ALUamux(ext5\_out,rs\_out,NPC,hi\_mux\_out,lo\_mux\_out,32'b0,32'b0,32'b0,ALUaMux\_control,aluamux\_out);

selector41 ALUbmux(rt\_out,ext16\_out,sext16\_out,32'b0,ALUbMux\_control,alubmux\_out);

HILO hilo\_unit(clk,rst,Z2[63:32],Z2[31:0],HI\_W,LO\_W,hi\_out,lo\_out);

selector41 RSmux(rs\_to\_mux,alu\_out,ALUo1,ALUo2,rsoutMux,rs\_out);

selector41 RTmux(rt\_to\_mux,alu\_out,ALUo1,ALUo2,rtoutMux,rt\_out);

selector41 HImux(hi\_out,z[63:32],Z1[63:32],Z2[63:32],hioutMux,hi\_mux\_out);

selector41 LOmux(lo\_out,z[31:0],Z1[31:0],Z2[31:0],looutMux,lo\_mux\_out);

//id-ex

always @ (posedge clk or posedge rst)

begin

if (rst==1 || (stall[1]==1 && stall[2]==0))

begin

id\_ex\_RF\_W<=0;

id\_ex\_aluc<=0;

id\_ex\_DM\_W<=0;

id\_ex\_rdMux<=0;

id\_ex\_rdcMux<=0;

ALUa<=0;

ALUb<=0;

Rdata1<=0;

IR2<=32'hffffffff;

end

else if (stall[1]==0)

begin

id\_ex\_RF\_W<=RF\_W\_control;

id\_ex\_aluc<=ALUC\_control;

id\_ex\_DM\_W<=DM\_W\_control;

id\_ex\_rdMux<=rdMux\_control;

id\_ex\_rdcMux<=rdcMux\_control;

ALUa<=aluamux\_out;

ALUb<=alubmux\_out;

Rdata1<=rt\_out;

IR2<=IR1;

pc2<=pc1;

end

else

begin

IR2<=32'h12345678;

end

end

//ex

alu alu\_unit(ALUa,ALUb,id\_ex\_aluc,rst,alu\_out,ex\_me\_Z,ex\_me\_C,ex\_me\_N,ex\_me\_O);

MULT mult\_unit(clk,rst,ALUa,ALUb,z);

//ex-me

always @ (posedge clk or posedge rst)

begin

if (rst || (stall[2] && !stall[3]))

begin

ex\_me\_RF\_W<=0;

ex\_me\_DM\_W<=0;

ex\_me\_rdMux<=0;

ex\_me\_rdcMux<=0;

Z1<=0;

ALUo1<=0;

Rdata2<=0;

IR3<=32'hffffffff;

end

else if (!stall[2])

begin

/\*对add，sub，addi三条指令的溢出（Overflow）情况做特殊处理（不写入）（改变RF\_W）\*/

if((IR2[31:26]==6'b000000&&IR2[5:0]==6'b100000&&ex\_me\_O==1)||

(IR2[31:26]==6'b000000&&IR2[5:0]==6'b100010&&ex\_me\_O==1)||

(IR2[31:26]==6'b001000&&ex\_me\_O==1))

begin

ex\_me\_RF\_W<=0;

end

else

ex\_me\_RF\_W<=id\_ex\_RF\_W;

ex\_me\_DM\_W<=id\_ex\_DM\_W;

ex\_me\_rdMux<=id\_ex\_rdMux;

ex\_me\_rdcMux<=id\_ex\_rdcMux;

Z1<=z;

ALUo1<=alu\_out;

Rdata2<=Rdata1;

IR3<=IR2;

pc3<=pc2;

end

end

//me

assign dmem\_data\_in=Rdata2;

assign dmem\_addr\_in=ALUo1;

assign dmem\_DM\_W=ex\_me\_DM\_W;

//me-wb

always @ (posedge clk or posedge rst)

begin

if (rst || (stall[3] && !stall[4]))

begin

me\_wb\_RF\_W<=0;

me\_wb\_rdMux<=0;

me\_wb\_rdcMux<=0;

me\_wb\_Z<=0;

me\_wb\_C<=0;

me\_wb\_N<=0;

me\_wb\_O<=0;

Wdata<=0;

Z2<=0;

ALUo2<=0;

IR4<=32'hffffffff;

end

else if (!stall[3])

begin

me\_wb\_RF\_W<=ex\_me\_RF\_W;

me\_wb\_rdMux<=ex\_me\_rdMux;

me\_wb\_rdcMux<=ex\_me\_rdcMux;

me\_wb\_Z<=ex\_me\_Z;

me\_wb\_C<=ex\_me\_C;

me\_wb\_N<=ex\_me\_N;

me\_wb\_O<=ex\_me\_O;

Wdata<=dmem\_out;

Z2<=Z1;

ALUo2<=ALUo1;

IR4<=IR3;

pc4<=pc3;

end

end

//wb

selector21 rdmux(ALUo2,Wdata,me\_wb\_rdMux,rdmux\_out);

selector41\_5 rdcmux(IR4[15:11],IR4[20:16],5'b11111,5'b0,me\_wb\_rdcMux,rdcmux\_out);

endmodule

module Divider(

input I\_CLK,

input rst,

output reg O\_CLK

);

parameter MAX\_HZ=100000;

reg [64:0]i=0;

initial

begin

O\_CLK<=0;

end

always @( posedge I\_CLK or posedge rst)

begin

if(rst==1)

begin

i<=0;

O\_CLK<=0;

end

else if(i==MAX\_HZ-1)

begin

i<=0;

O\_CLK<=~O\_CLK;

end

else

begin

i<=i+1;

end

end

endmodule

module DMEM(

input clk,

input ena,

input wena,//高电平写入有效，低电平读有效

input [10:0] addr,

input [31:0] data\_in,

output reg [31:0] data\_out

);

reg [31:0] memory [0:31];//reg数组,前面为位数，后面为数组元素个数

always @(\*)

begin

if(ena==0)

data\_out=32'hzzzzzzzz;

else

begin

if(wena==1)//写有效

begin

// memory[addr]=data\_in;

end

else//读有效

begin

data\_out=memory[addr];

end

end

end

always @(posedge clk)

begin

if(ena==0)

begin

end

// data\_out=32'hzzzzzzzz;

else

begin

if(wena==1)//写有效

begin

memory[addr]=data\_in;

end

else//读有效

begin

// data\_out=memory[addr];

end

end

end

endmodule

module equal\_unit(

input [31:0] a,

input [31:0] b,

output c

);

assign c=(a==b)?1:0;

endmodule

module EXT\_5(

input [4:0]a,

output [31:0]b

);

assign b={{27{1'b0}},a};

endmodule

module EXT\_16(

input [15:0]a,

output [31:0]b

);

assign b={{16{1'b0}},a};

endmodule

module S\_EXT\_16(

input [15:0]a,

output [31:0]b

);

assign b={{16{a[15]}},a};

endmodule

module S\_EXT\_18(

input [17:0]a,

output [31:0]b

);

assign b={{14{a[17]}},a};

endmodule

module IMEM(

input [10:0]addr,

output [31:0]instr

);

imem\_ip im(.a(addr),.spo(instr));

endmodule

module joint\_pc\_im(

input [27:0] imem\_in,

input [3:0] pc\_in,

output [31:0] joint\_out

);

assign joint\_out={pc\_in,imem\_in};

endmodule

module selector41(

input [31:0] iC0,

input [31:0] iC1,

input [31:0] iC2,

input [31:0] iC3,

input [1:0]iS,

output reg [31:0] oZ

);

always @(\*)

begin

if(iS==0)

oZ=iC0;

else if(iS==1)

oZ=iC1;

else if(iS==2)

oZ=iC2;

else

oZ=iC3;

end

endmodule

module selector81(

input [31:0] iC0,

input [31:0] iC1,

input [31:0] iC2,

input [31:0] iC3,

input [31:0] iC4,

input [31:0] iC5,

input [31:0] iC6,

input [31:0] iC7,

input [2:0]iS,

output reg [31:0] oZ

);

always @(\*)

begin

if(iS==0)

oZ=iC0;

else if(iS==1)

oZ=iC1;

else if(iS==2)

oZ=iC2;

else if(iS==3)

oZ=iC3;

else if(iS==4)

oZ=iC4;

else if(iS==5)

oZ=iC5;

else if(iS==6)

oZ=iC6;

else

oZ=iC7;

end

endmodule

module selector41\_5(

input [4:0] iC0,

input [4:0] iC1,

input [4:0] iC2,

input [4:0] iC3,

input [1:0]iS,

output reg [4:0] oZ

);

always @(\*)

begin

if(iS==0)

oZ=iC0;

else if(iS==1)

oZ=iC1;

else if(iS==2)

oZ=iC2;

else

oZ=iC3;

end

endmodule

module selector21(

input [31:0] iC0,

input [31:0] iC1,

input iS,

output reg [31:0] oZ

);

always @(\*)

begin

if(iS==0)

oZ=iC0;

else

oZ=iC1;

end

endmodule

module sccomp\_dataflow(

input clk\_in,

input reset,

output [7:0] o\_seg,

output [7:0] o\_sel

/\*output clk1,

output [31:0] inst,

output [31:0] pc,

output [31:0] dmem\_out,

output [31:0] imem\_addr,

output [31:0] dmem\_data,

output [31:0] dmem\_addr,

output dmem\_DM\_W,

output [31:0] add\_out,

output equal\_out,

output [31:0] join\_out,

output [31:0] pcmux\_out,

output [31:0] aluamux\_out,

output [31:0] alubmux\_out,

output [31:0] rdmux\_out,

output [4:0] rdcmux\_out,

output [31:0] ext5\_out,

output [31:0] ext16\_out,

output [31:0] sext16\_out,

output [31:0] sext18\_out,

output [31:0] rs\_out,

output [31:0] rt\_out,

output [31:0] alu\_out,

output RF\_W\_control,

output [3:0] ALUC\_control,

output [1:0] PCMux\_control,

output [2:0] ALUaMux\_control,

output [1:0] ALUbMux\_control,

output rdMux\_control,

output [1:0] rdcMux\_control,

output CS\_control,

output DM\_R\_control,

output DM\_W\_control,

output [4:0] stall,

output HI\_W,

output LO\_W,

output [1:0] rsoutMux,

output [1:0] rtoutMux,

output [1:0] hioutMux,

output [1:0] looutMux,

output [31:0] rs\_to\_mux,

output [31:0] rt\_to\_mux,

output [31:0] hi\_out,

output [31:0] lo\_out,

output [31:0] hi\_mux\_out,

output [31:0] lo\_mux\_out,

output [63:0] z,

output [63:0] Z1,

output [63:0] Z2,

output id\_ex\_RF\_W,

output [3:0] id\_ex\_aluc,

output id\_ex\_DM\_W,

output id\_ex\_rdMux,

output [1:0] id\_ex\_rdcMux,

output ex\_me\_RF\_W,

output ex\_me\_DM\_W,

output ex\_me\_rdMux,

output [1:0] ex\_me\_rdcMux,

output ex\_me\_Z,

output ex\_me\_C,

output ex\_me\_N,

output ex\_me\_O,

output me\_wb\_RF\_W,

output me\_wb\_rdMux,

output [1:0] me\_wb\_rdcMux,

output me\_wb\_Z,

output me\_wb\_C,

output me\_wb\_N,

output me\_wb\_O,

output [31:0] NPC,

output [31:0] IR1,

output [31:0] IR2,

output [31:0] IR3,

output [31:0] IR4,

output [31:0] ALUa,

output [31:0] ALUb,

output [31:0] ALUo1,

output [31:0] ALUo2,

output [31:0] Rdata1,

output [31:0] Rdata2,

output [31:0] Wdata,

output [31:0] pc1,

output [31:0] pc2,

output [31:0] pc3,

output [31:0] pc4,

output [31:0] out\_reg\*/

);

wire clk1;

wire [31:0] inst;

wire [31:0] pc;

wire [31:0] dmem\_out;

wire [31:0] imem\_addr;

wire [31:0] dmem\_data;

wire [31:0] dmem\_addr;

wire dmem\_DM\_W;

wire [31:0] add\_out;

wire equal\_out;

wire [31:0] join\_out;

wire [31:0] pcmux\_out;

wire [31:0] aluamux\_out;

wire [31:0] alubmux\_out;

wire [31:0] rdmux\_out;

wire [4:0] rdcmux\_out;

wire [31:0] ext5\_out;

wire [31:0] ext16\_out;

wire [31:0] sext16\_out;

wire [31:0] sext18\_out;

wire [31:0] rs\_out;

wire [31:0] rt\_out;

wire [31:0] alu\_out;

wire RF\_W\_control;

wire [3:0] ALUC\_control;

wire [1:0] PCMux\_control;

wire [2:0] ALUaMux\_control;

wire [1:0] ALUbMux\_control;

wire rdMux\_control;

wire [1:0] rdcMux\_control;

wire CS\_control;

wire DM\_R\_control;

wire DM\_W\_control;

wire [4:0] stall;

wire HI\_W;

wire LO\_W;

wire [1:0] rsoutMux;

wire [1:0] rtoutMux;

wire [1:0] hioutMux;

wire [1:0] looutMux;

wire [31:0] rs\_to\_mux;

wire [31:0] rt\_to\_mux;

wire [31:0] hi\_out;

wire [31:0] lo\_out;

wire [31:0] hi\_mux\_out;

wire [31:0] lo\_mux\_out;

wire [63:0] z;

wire [63:0] Z1;

wire [63:0] Z2;

wire id\_ex\_RF\_W;

wire [3:0] id\_ex\_aluc;

wire id\_ex\_DM\_W;

wire id\_ex\_rdMux;

wire [1:0] id\_ex\_rdcMux;

wire ex\_me\_RF\_W;

wire ex\_me\_DM\_W;

wire ex\_me\_rdMux;

wire [1:0] ex\_me\_rdcMux;

wire ex\_me\_Z;

wire ex\_me\_C;

wire ex\_me\_N;

wire ex\_me\_O;

wire me\_wb\_RF\_W;

wire me\_wb\_rdMux;

wire [1:0] me\_wb\_rdcMux;

wire me\_wb\_Z;

wire me\_wb\_C;

wire me\_wb\_N;

wire me\_wb\_O;

wire [31:0] NPC;

wire [31:0] IR1;

wire [31:0] IR2;

wire [31:0] IR3;

wire [31:0] IR4;

wire [31:0] ALUa;

wire [31:0] ALUb;

wire [31:0] ALUo1;

wire [31:0] ALUo2;

wire [31:0] Rdata1;

wire [31:0] Rdata2;

wire [31:0] Wdata;

wire [31:0] pc1;

wire [31:0] pc2;

wire [31:0] pc3;

wire [31:0] pc4;

wire [31:0] out\_reg;

wire [31:0] dm\_addr;

wire [31:0] im\_addr;

assign im\_addr = imem\_addr- 32'h00400000;

assign dm\_addr = dmem\_addr- 32'h10010000;

IMEM imemory(pc[12:2],inst);

DMEM dmemory(clk1,1'b1,dmem\_DM\_W,dm\_addr[12:2],dmem\_data,dmem\_out);

/\* module cpu(

input clk,

input rst,

input [31:0] imem\_out,

input [31:0] dmem\_out,

output [31:0] imem\_in,

output [31:0] dmem\_data\_in,

output [31:0] dmem\_addr\_in,

output dmem\_DM\_W

);\*/

cpu sccpu(clk1,reset,inst,dmem\_out,imem\_addr,dmem\_data,dmem\_addr,dmem\_DM\_W,pc,

add\_out,

equal\_out,

join\_out,

pcmux\_out,

aluamux\_out,

alubmux\_out,

rdmux\_out,

rdcmux\_out,

ext5\_out,

ext16\_out,

sext16\_out,

sext18\_out,

rs\_out,

rt\_out,

alu\_out,

RF\_W\_control,

ALUC\_control,

PCMux\_control,

ALUaMux\_control,

ALUbMux\_control,

rdMux\_control,

rdcMux\_control,

CS\_control,

DM\_R\_control,

DM\_W\_control,

stall,

HI\_W,

LO\_W,

rsoutMux,

rtoutMux,

hioutMux,

looutMux,

rs\_to\_mux,

rt\_to\_mux,

hi\_out,

lo\_out,

hi\_mux\_out,

lo\_mux\_out,

z,

Z1,

Z2,

id\_ex\_RF\_W,

id\_ex\_aluc,

id\_ex\_DM\_W,

id\_ex\_rdMux,

id\_ex\_rdcMux,

ex\_me\_RF\_W,

ex\_me\_DM\_W,

ex\_me\_rdMux,

ex\_me\_rdcMux,

ex\_me\_Z,

ex\_me\_C,

ex\_me\_N,

ex\_me\_O,

me\_wb\_RF\_W,

me\_wb\_rdMux,

me\_wb\_rdcMux,

me\_wb\_Z,

me\_wb\_C,

me\_wb\_N,

me\_wb\_O,

NPC,

IR1,

IR2,

IR3,

IR4,

ALUa,

ALUb,

ALUo1,

ALUo2,

Rdata1,

Rdata2,

Wdata,

pc1,pc2,pc3,pc4,out\_reg);

//assign clk1=clk\_in;

Divider d1(clk\_in,reset,clk1);

seg7x16 s716(clk\_in,reset,1,out\_reg,o\_seg,o\_sel);

endmodule

`timescale 1ns / 1ns

module test(

);

reg clk, rst;

wire clk1;

wire [31:0] inst, pc;

wire [31:0] dmem\_out;

wire [31:0] imem\_addr;

wire [31:0] dmem\_data;

wire [31:0] dmem\_addr;

wire dmem\_DM\_W;

wire [31:0] add\_out;

wire equal\_out;

wire [31:0] join\_out;

wire [31:0] pcmux\_out;

wire [31:0] aluamux\_out;

wire [31:0] alubmux\_out;

wire [31:0] rdmux\_out;

wire [4:0] rdcmux\_out;

wire [31:0] ext5\_out;

wire [31:0] ext16\_out;

wire [31:0] sext16\_out;

wire [31:0] sext18\_out;

wire [31:0] rs\_out;

wire [31:0] rt\_out;

wire [31:0] alu\_out;

/\*控制信号\*/

/\*---------------------------------\*/

wire RF\_W\_control;

wire [3:0] ALUC\_control;//alu识别码

/\*选择器\*/

wire [1:0] PCMux\_control;

wire [2:0] ALUaMux\_control;

wire [1:0] ALUbMux\_control;

wire rdMux\_control;

wire [1:0] rdcMux\_control;

/\*dmem信号\*/

wire CS\_control;//无用

wire DM\_R\_control;//无用

wire DM\_W\_control;

wire [4:0] stall;

//wire [4:0] stall1;

wire HI\_W;

wire LO\_W;

wire [1:0] rsoutMux;

wire [1:0] rtoutMux;

wire [1:0] hioutMux;

wire [1:0] looutMux;

wire [31:0] rs\_to\_mux;

wire [31:0] rt\_to\_mux;

wire [31:0] hi\_out;

wire [31:0] lo\_out;

wire [31:0] hi\_mux\_out;

wire [31:0] lo\_mux\_out;

wire [63:0] z;

wire [63:0] Z1;

wire [63:0] Z2;

/\*id-ex段间控制信号寄存器\*/

wire id\_ex\_RF\_W;

wire [3:0] id\_ex\_aluc;

wire id\_ex\_DM\_W;

wire id\_ex\_rdMux;

wire [1:0] id\_ex\_rdcMux;

/\*ex-me段间控制信号寄存器\*/

wire ex\_me\_RF\_W;

// wire [3:0] ex\_me\_aluc;

wire ex\_me\_DM\_W;

wire ex\_me\_rdMux;

wire [1:0] ex\_me\_rdcMux;

wire ex\_me\_Z;

wire ex\_me\_C;

wire ex\_me\_N;

wire ex\_me\_O;

/\*me-wb段间控制信号寄存器\*/

wire me\_wb\_RF\_W;

// wire [3:0] me\_wb\_aluc;

//wire me\_wb\_DM\_W;

wire me\_wb\_rdMux;

wire [1:0] me\_wb\_rdcMux;

wire me\_wb\_Z;

wire me\_wb\_C;

wire me\_wb\_N;

wire me\_wb\_O;

/\*---------------------------------\*/

/\*段间寄存器\*/

wire [31:0] NPC;

wire [31:0] IR1;

wire [31:0] IR2;

wire [31:0] IR3;

wire [31:0] IR4;

wire [31:0] ALUa;

wire [31:0] ALUb;

wire [31:0] ALUo1;

wire [31:0] ALUo2;

wire [31:0] Rdata1;

wire [31:0] Rdata2;

wire [31:0] Wdata;

wire [31:0] pc1;

wire [31:0] pc2;

wire [31:0] pc3;

wire [31:0] pc4;

wire [31:0] out\_reg;

integer file\_output;

integer counter=0 ;

initial begin

file\_output = $fopen("result.txt", "a");

clk = 1'b0;

rst = 1'b1;

#50 rst=1'b0;

end

always begin

#20 clk = !clk;

end

always @(posedge clk) begin

if(counter>=1200)

begin

$fclose(file\_output);

end

else if(clk==1'b1&&rst==1'b0)

begin

if(IR4!=32'hffffffff)

begin

counter=counter+1;

/\* $fdisplay(file\_output, "pc: %h", pc4);

$fdisplay(file\_output, "instr: %h", IR4);

$fdisplay(file\_output, "regfile0: %h", test.uut.sccpu.cpu\_ref.array\_reg[0]);

$fdisplay(file\_output, "regfile1: %h", test.uut.sccpu.cpu\_ref.array\_reg[1]);

$fdisplay(file\_output, "regfile2: %h", test.uut.sccpu.cpu\_ref.array\_reg[2]);

$fdisplay(file\_output, "regfile3: %h", test.uut.sccpu.cpu\_ref.array\_reg[3]);

$fdisplay(file\_output, "regfile4: %h", test.uut.sccpu.cpu\_ref.array\_reg[4]);

$fdisplay(file\_output, "regfile5: %h", test.uut.sccpu.cpu\_ref.array\_reg[5]);

$fdisplay(file\_output, "regfile6: %h", test.uut.sccpu.cpu\_ref.array\_reg[6]);

$fdisplay(file\_output, "regfile7: %h", test.uut.sccpu.cpu\_ref.array\_reg[7]);

$fdisplay(file\_output, "regfile8: %h", test.uut.sccpu.cpu\_ref.array\_reg[8]);

$fdisplay(file\_output, "regfile9: %h", test.uut.sccpu.cpu\_ref.array\_reg[9]);

$fdisplay(file\_output, "regfile10: %h", test.uut.sccpu.cpu\_ref.array\_reg[10]);

$fdisplay(file\_output, "regfile11: %h", test.uut.sccpu.cpu\_ref.array\_reg[11]);

$fdisplay(file\_output, "regfile12: %h", test.uut.sccpu.cpu\_ref.array\_reg[12]);

$fdisplay(file\_output, "regfile13: %h", test.uut.sccpu.cpu\_ref.array\_reg[13]);

$fdisplay(file\_output, "regfile14: %h", test.uut.sccpu.cpu\_ref.array\_reg[14]);

$fdisplay(file\_output, "regfile15: %h", test.uut.sccpu.cpu\_ref.array\_reg[15]);

$fdisplay(file\_output, "regfile16: %h", test.uut.sccpu.cpu\_ref.array\_reg[16]);

$fdisplay(file\_output, "regfile17: %h", test.uut.sccpu.cpu\_ref.array\_reg[17]);

$fdisplay(file\_output, "regfile18: %h", test.uut.sccpu.cpu\_ref.array\_reg[18]);

$fdisplay(file\_output, "regfile19: %h", test.uut.sccpu.cpu\_ref.array\_reg[19]);

$fdisplay(file\_output, "regfile20: %h", test.uut.sccpu.cpu\_ref.array\_reg[20]);

$fdisplay(file\_output, "regfile21: %h", test.uut.sccpu.cpu\_ref.array\_reg[21]);

$fdisplay(file\_output, "regfile22: %h", test.uut.sccpu.cpu\_ref.array\_reg[22]);

$fdisplay(file\_output, "regfile23: %h", test.uut.sccpu.cpu\_ref.array\_reg[23]);

$fdisplay(file\_output, "regfile24: %h", test.uut.sccpu.cpu\_ref.array\_reg[24]);

$fdisplay(file\_output, "regfile25: %h", test.uut.sccpu.cpu\_ref.array\_reg[25]);

$fdisplay(file\_output, "regfile26: %h", test.uut.sccpu.cpu\_ref.array\_reg[26]);

$fdisplay(file\_output, "regfile27: %h", test.uut.sccpu.cpu\_ref.array\_reg[27]);

$fdisplay(file\_output, "regfile28: %h", test.uut.sccpu.cpu\_ref.array\_reg[28]);

$fdisplay(file\_output, "regfile29: %h", test.uut.sccpu.cpu\_ref.array\_reg[29]);

$fdisplay(file\_output, "regfile30: %h", test.uut.sccpu.cpu\_ref.array\_reg[30]);

$fdisplay(file\_output, "regfile31: %h", test.uut.sccpu.cpu\_ref.array\_reg[31]);\*/

end

end

else

begin

end

end

sccomp\_dataflow uut(

.clk\_in(clk),

.reset(rst),

.clk1(clk1),

.inst(inst),

.pc(pc),

.dmem\_out(dmem\_out),

.imem\_addr(imem\_addr),

.dmem\_data(dmem\_data),

.dmem\_addr(dmem\_addr),

.dmem\_DM\_W(dmem\_DM\_W),

.add\_out(add\_out),

.equal\_out(equal\_out),

.join\_out(join\_out),

.pcmux\_out(pcmux\_out),

.aluamux\_out(aluamux\_out),

.alubmux\_out(alubmux\_out),

.rdmux\_out(rdmux\_out),

.rdcmux\_out(rdcmux\_out),

.ext5\_out(ext5\_out),

.ext16\_out(ext16\_out),

.sext16\_out(sext16\_out),

.sext18\_out(sext18\_out),

.rs\_out(rs\_out),

.rt\_out(rt\_out),

.alu\_out(alu\_out),

.RF\_W\_control(RF\_W\_control),

.ALUC\_control(ALUC\_control),

.PCMux\_control(PCMux\_control),

.ALUaMux\_control(ALUaMux\_control),

.ALUbMux\_control(ALUbMux\_control),

.rdMux\_control(rdMux\_control),

.rdcMux\_control(rdcMux\_control),

.CS\_control(CS\_control),

.DM\_R\_control(DM\_R\_control),

.DM\_W\_control(DM\_W\_control),

.stall(stall),

.HI\_W(HI\_W),

.LO\_W(LO\_W),

.rsoutMux(rsoutMux),

.rtoutMux(rtoutMux),

.hioutMux(hioutMux),

.looutMux(looutMux),

.rs\_to\_mux(rs\_to\_mux),

.rt\_to\_mux(rt\_to\_mux),

.hi\_out(hi\_out),

.lo\_out(lo\_out),

.hi\_mux\_out(hi\_mux\_out),

.lo\_mux\_out(lo\_mux\_out),

.z(z),

.Z1(Z1),

.Z2(Z2),

.id\_ex\_RF\_W(id\_ex\_RF\_W),

.id\_ex\_aluc(id\_ex\_aluc),

.id\_ex\_DM\_W(id\_ex\_DM\_W),

.id\_ex\_rdMux(id\_ex\_rdMux),

.id\_ex\_rdcMux(id\_ex\_rdcMux),

.ex\_me\_RF\_W(ex\_me\_RF\_W),

.ex\_me\_DM\_W(ex\_me\_DM\_W),

.ex\_me\_rdMux(ex\_me\_rdMux),

.ex\_me\_rdcMux(ex\_me\_rdcMux),

.ex\_me\_Z(ex\_me\_Z),

.ex\_me\_C(ex\_me\_C),

.ex\_me\_N(ex\_me\_N),

.ex\_me\_O(ex\_me\_O),

.me\_wb\_RF\_W(me\_wb\_RF\_W),

.me\_wb\_rdMux(me\_wb\_rdMux),

.me\_wb\_rdcMux(me\_wb\_rdcMux),

.me\_wb\_Z(me\_wb\_Z),

.me\_wb\_C(me\_wb\_C),

.me\_wb\_N(me\_wb\_N),

.me\_wb\_O(me\_wb\_O),

.NPC(NPC),

.IR1(IR1),

.IR2(IR2),

.IR3(IR3),

.IR4(IR4),

.ALUa(ALUa),

.ALUb(ALUb),

.ALUo1(ALUo1),

.ALUo2(ALUo2),

.Rdata1(Rdata1),

.Rdata2(Rdata2),

.Wdata(Wdata),

.pc1(pc1),.pc2(pc2),.pc3(pc3),.pc4(pc4),

.out\_reg(out\_reg)

);

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 13:00:18 05/03/2017

// Design Name:

// Module Name: seg7x16

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module seg7x16(

input clk,

input reset,

input cs,

input [31:0] i\_data,

output [7:0] o\_seg,

output [7:0] o\_sel

);

reg [14:0] cnt;

always @ (posedge clk, posedge reset)

if (reset)

cnt <= 0;

else

cnt <= cnt + 1'b1;

wire seg7\_clk = cnt[14];

reg [2:0] seg7\_addr;

always @ (posedge seg7\_clk, posedge reset)

if(reset)

seg7\_addr <= 0;

else

seg7\_addr <= seg7\_addr + 1'b1;

reg [7:0] o\_sel\_r;

always @ (\*)

case(seg7\_addr)

7 : o\_sel\_r = 8'b01111111;

6 : o\_sel\_r = 8'b10111111;

5 : o\_sel\_r = 8'b11011111;

4 : o\_sel\_r = 8'b11101111;

3 : o\_sel\_r = 8'b11110111;

2 : o\_sel\_r = 8'b11111011;

1 : o\_sel\_r = 8'b11111101;

0 : o\_sel\_r = 8'b11111110;

endcase

reg [31:0] i\_data\_store;

always @ (posedge clk, posedge reset)

if(reset)

i\_data\_store <= 0;

else if(cs)

i\_data\_store <= i\_data;

reg [7:0] seg\_data\_r;

always @ (\*)

case(seg7\_addr)

0 : seg\_data\_r = i\_data\_store[3:0];

1 : seg\_data\_r = i\_data\_store[7:4];

2 : seg\_data\_r = i\_data\_store[11:8];

3 : seg\_data\_r = i\_data\_store[15:12];

4 : seg\_data\_r = i\_data\_store[19:16];

5 : seg\_data\_r = i\_data\_store[23:20];

6 : seg\_data\_r = i\_data\_store[27:24];

7 : seg\_data\_r = i\_data\_store[31:28];

endcase

reg [7:0] o\_seg\_r;

always @ (posedge clk, posedge reset)

if(reset)

o\_seg\_r <= 8'hff;

else

case(seg\_data\_r)

4'h0 : o\_seg\_r <= 8'hC0;

4'h1 : o\_seg\_r <= 8'hF9;

4'h2 : o\_seg\_r <= 8'hA4;

4'h3 : o\_seg\_r <= 8'hB0;

4'h4 : o\_seg\_r <= 8'h99;

4'h5 : o\_seg\_r <= 8'h92;

4'h6 : o\_seg\_r <= 8'h82;

4'h7 : o\_seg\_r <= 8'hF8;

4'h8 : o\_seg\_r <= 8'h80;

4'h9 : o\_seg\_r <= 8'h90;

4'hA : o\_seg\_r <= 8'h88;

4'hB : o\_seg\_r <= 8'h83;

4'hC : o\_seg\_r <= 8'hC6;

4'hD : o\_seg\_r <= 8'hA1;

4'hE : o\_seg\_r <= 8'h86;

4'hF : o\_seg\_r <= 8'h8E;

endcase

assign o\_sel = o\_sel\_r;

assign o\_seg = o\_seg\_r;

endmodule

module HILO(

input clk,

input rst,

input [31:0]hi\_in,

input [31:0]lo\_in,

input hi\_w,

input lo\_w,

output [31:0]hi\_out,

output [31:0]lo\_out

);

reg [31:0]hi;

reg [31:0]lo;

assign hi\_out=hi;

assign lo\_out=lo;

always @(negedge clk or posedge rst)

begin

if(rst==1'b1)

begin

hi<=32'b0;

lo<=32'b0;

end

else

begin

if(hi\_w==1)

begin

hi<=hi\_in;

end

else

begin

end

if(lo\_w==1)

begin

lo<=lo\_in;

end

else

begin

end

end

end

endmodule

module MULT(

input clk,

input reset,

input [31:0]a,

input [31:0]b,

output [63:0]z

);

reg [63:0] temp;

reg [63:0] stored0;

reg [63:0] stored1;

reg [63:0] stored2;

reg [63:0] stored3;

reg [63:0] stored4;

reg [63:0] stored5;

reg [63:0] stored6;

reg [63:0] stored7;

reg [63:0] stored8;

reg [63:0] stored9;

reg [63:0] stored10;

reg [63:0] stored11;

reg [63:0] stored12;

reg [63:0] stored13;

reg [63:0] stored14;

reg [63:0] stored15;

reg [63:0] stored16;

reg [63:0] stored17;

reg [63:0] stored18;

reg [63:0] stored19;

reg [63:0] stored20;

reg [63:0] stored21;

reg [63:0] stored22;

reg [63:0] stored23;

reg [63:0] stored24;

reg [63:0] stored25;

reg [63:0] stored26;

reg [63:0] stored27;

reg [63:0] stored28;

reg [63:0] stored29;

reg [63:0] stored30;

reg [63:0] stored31;

/\*---------------------------\*/

reg [63:0] add01;

reg [63:0] add23;

reg [63:0] add45;

reg [63:0] add67;

reg [63:0] add89;

reg [63:0] add1011;

reg [63:0] add1213;

reg [63:0] add1415;

reg [63:0] add1617;

reg [63:0] add1819;

reg [63:0] add2021;

reg [63:0] add2223;

reg [63:0] add2425;

reg [63:0] add2627;

reg [63:0] add2829;

reg [63:0] add3031;

/\*--------------------------\*/

reg [63:0] add0123;

reg [63:0] add4567;

reg [63:0] add891011;

reg [63:0] add12131415;

reg [63:0] add16171819;

reg [63:0] add20212223;

reg [63:0] add24252627;

reg [63:0] add28293031;

/\*-----------------------------\*/

reg [63:0] add01234567;

reg [63:0] add89101112131415;

reg [63:0] add1617181920212223;

reg [63:0] add2425262728293031;

/\*-----------------------------\*/

reg [63:0] add0\_15;

reg [63:0] add16\_31;

/\*------------------------------\*/

// reg [63:0] add0\_31;

wire [31:0] aa;

wire [31:0] bb;

assign aa=a[31]?(~(a-1)) :a;

assign bb=b[31]?(~(b-1)) :b;

always @(\*)

begin

if(reset)

begin

temp=0;

stored0 = 0;

stored1 = 0;

stored2 = 0;

stored3 = 0;

stored4 = 0;

stored5 = 0;

stored6 = 0;

stored7 = 0;

stored8 = 0;

stored9 = 0;

stored10 = 0;

stored11 = 0;

stored12 = 0;

stored13 = 0;

stored14 = 0;

stored15 = 0;

stored16 = 0;

stored17 = 0;

stored18 = 0;

stored19 = 0;

stored20 = 0;

stored21 = 0;

stored22 = 0;

stored23 = 0;

stored24 = 0;

stored25 = 0;

stored26 = 0;

stored27 = 0;

stored28 = 0;

stored29 = 0;

stored30 = 0;

stored31 = 0;

add01 = 0;

add23 = 0;

add45 = 0;

add67 = 0;

add89 = 0;

add1011 = 0;

add1213 = 0;

add1415 = 0;

add1617 = 0;

add1819 = 0;

add2021 = 0;

add2223 = 0;

add2425 = 0;

add2627 = 0;

add2829 = 0;

add3031 = 0;

add0123 = 0;

add4567 = 0;

add891011 = 0;

add12131415 = 0;

add16171819 = 0;

add20212223 = 0;

add24252627 = 0;

add28293031 = 0;

add01234567 = 0;

add89101112131415 = 0;

add1617181920212223 = 0;

add2425262728293031 = 0;

add0\_15 = 0;

add16\_31 = 0;

//add0\_31 = 0;

end

else

begin

stored0 = bb[0] ? {32'b0 , aa} : 64'b0;

stored1 = bb[1] ? {31'b0 , aa , 1'b0} : 64'b0;

stored2 = bb[2] ? {30'b0 , aa , 2'b0} : 64'b0;

stored3 = bb[3] ? {29'b0 , aa , 3'b0} : 64'b0;

stored4 = bb[4] ? {28'b0 , aa , 4'b0} : 64'b0;

stored5 = bb[5] ? {27'b0 , aa , 5'b0} : 64'b0;

stored6 = bb[6] ? {26'b0 , aa , 6'b0} : 64'b0;

stored7 = bb[7] ? {25'b0 , aa , 7'b0} : 64'b0;

stored8 = bb[8] ? {24'b0 , aa , 8'b0} : 64'b0;

stored9 = bb[9] ? {23'b0 , aa , 9'b0} : 64'b0;

stored10 = bb[10] ? {22'b0 , aa , 10'b0} : 64'b0;

stored11 = bb[11] ? {21'b0 , aa , 11'b0} : 64'b0;

stored12 = bb[12] ? {20'b0 , aa , 12'b0} : 64'b0;

stored13 = bb[13] ? {19'b0 , aa , 13'b0} : 64'b0;

stored14 = bb[14] ? {18'b0 , aa , 14'b0} : 64'b0;

stored15 = bb[15] ? {17'b0 , aa , 15'b0} : 64'b0;

stored16 = bb[16] ? {16'b0 , aa , 16'b0} : 64'b0;

stored17 = bb[17] ? {15'b0 , aa , 17'b0} : 64'b0;

stored18 = bb[18] ? {14'b0 , aa , 18'b0} : 64'b0;

stored19 = bb[19] ? {13'b0 , aa , 19'b0} : 64'b0;

stored20 = bb[20] ? {12'b0 , aa , 20'b0} : 64'b0;

stored21 = bb[21] ? {11'b0 , aa , 21'b0} : 64'b0;

stored22 = bb[22] ? {10'b0 , aa , 22'b0} : 64'b0;

stored23 = bb[23] ? {9'b0 , aa , 23'b0} : 64'b0;

stored24 = bb[24] ? {8'b0 , aa , 24'b0} : 64'b0;

stored25 = bb[25] ? {7'b0 , aa , 25'b0} : 64'b0;

stored26 = bb[26] ? {6'b0 , aa , 26'b0} : 64'b0;

stored27 = bb[27] ? {5'b0 , aa , 27'b0} : 64'b0;

stored28 = bb[28] ? {4'b0 , aa , 28'b0} : 64'b0;

stored29 = bb[29] ? {3'b0 , aa , 29'b0} : 64'b0;

stored30 = bb[30] ? {2'b0 , aa , 30'b0} : 64'b0;

stored31 = bb[31] ? {1'b0 , aa , 31'b0} : 64'b0;

add01 = stored0 + stored1;

add23 = stored2 + stored3;

add45 = stored4 + stored5;

add67 = stored6 + stored7;

add89 = stored8 + stored9;

add1011 = stored10 + stored11;

add1213 = stored12 + stored13;

add1415 = stored14 + stored15;

add1617 = stored16 + stored17;

add1819 = stored18 + stored19;

add2021 = stored20 + stored21;

add2223 = stored22 + stored23;

add2425 = stored24 + stored25;

add2627 = stored26 + stored27;

add2829 = stored28 + stored29;

add3031 = stored30 + stored31;

add0123 = add01 + add23;

add4567 = add45 + add67;

add891011 = add89 + add1011;

add12131415 = add1213 + add1415;

add16171819 = add1617 + add1819;

add20212223 = add2021 + add2223;

add24252627 = add2425 + add2627;

add28293031 = add2829 + add3031;

add01234567 = add0123 + add4567;

add89101112131415 = add891011 + add12131415;

add1617181920212223 = add16171819 + add20212223;

add2425262728293031 = add24252627 + add28293031;

add0\_15 = add01234567 + add89101112131415;

add16\_31 = add1617181920212223 + add2425262728293031;

// add0\_31 = add0\_15 + add16\_31;

temp = add0\_15 + add16\_31;

end

end

assign z= (a[31]^b[31]) ?((~temp)+1): temp;

endmodule