

计算机组成原理实验报告

单周期流水线处理器开发

班级：1618001

学号：161810329

姓名：荀正

专业：培优班

2020.5.11

一、功能设计说明

1.完成的指令集：

addiu、slt、beq、addi、sw、sub、j、lw、ori、and、or、add、lui、xor

2.处理器为单周期设计

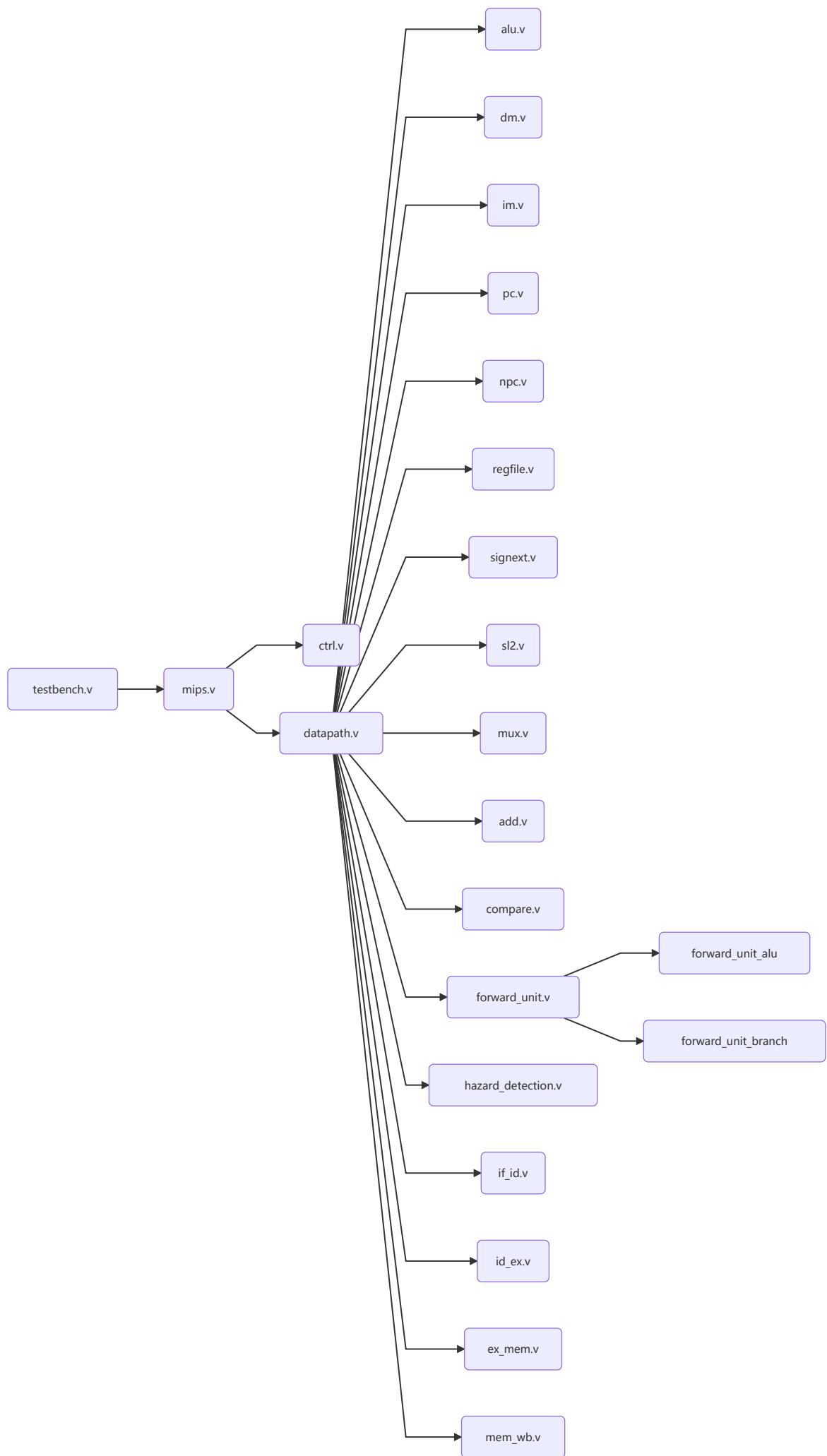
3.功能模块采用教材图设计，在控制信号上，为了应对一些可能出现的拓展，做了一些微调。

二、模块化和层次设计说明

模块对应

文件名	对应部件名
alu.v	算数逻辑模块
ctrl.v	控制器模块
datapath.v	数据通路模块
dm.v	数据存储模块 用来存取数据
im.v	指令存储模块 取指令
mips.v	进入的主模块
mux.v	选择器模块 复用
npc.v	计算下一个pc模块
pc.v	pc模块
regfile.v	寄存器模块
signext.v	符号拓展模块
add.v	与门模块
compare.v	比较模块
forward_unit.v	转发模块
hazard_detection.v	冒险检测模块
if_id.v	if_id流水寄存器模块
id_ex.v	id_ex流水寄存器模块
ex_mem.v	ex_mem流水寄存器模块
mem_wb.v	mem_wb流水寄存器模块
testbench.v	测试运行模块

关系图



三、模块定义

1、pc模块定义

1) 基本描述

输出当前指令地址并保存下一条指令地址。复位后，PC指向0x0000_3000，即第一条指令的地址。

2) 模块接口

信号名	方向	描述
NPC [31:2]	I	下条指令的地址
clk	I	时钟信号 上升沿触发
rst	I	复位信号 1: 复位 0: 无效
PC_write	I	是否写信号
PC [31:2]	O	30位指令存储器地址

3) 功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，PC设置为0x0000_3000
2	输出NPC	在上升沿的时候输出NPC
3	阻塞	在特定条件下不改变PC实现阻塞

2、npc模块定义

1) 基本描述

根据当前的指令地址结合分支、跳转等信号计算下一条指令地址并输出给PC

2) 模块接口

信号名	方向	描述
PC [31:0]	I	当前pc
instruction [25:0]	I	当前指令的后26位, j指令使用
beqInstruction [31:0]	I	根据指令计算得到beq指令地址
branch [1:0]	I	分支信号 2'b00: 无分支 2'b10: beq 2'b11: bne
jump[1:0]	I	跳转信号 2'b00: 无跳转 2'b01: j 2'b10: jr 2'b11: jal
zero	I	alu传来的zero信号, 用于branch指令 1: 相等 0: 不相等
NPC [31:2]	O	下一个指令地址
fourPC[31:2]	O	当前指令加四地址, 留作拓展

3) 功能定义

序号	功能名称	功能描述
1	输出NPC	计算下条指令并输出
2	输出PC+4	留给jal指令使用

3、im模块定义

1) 基本描述

im (instruction memory) 的主要功能是存储指令, 根据指令输出相应指令。

2) 模块接口

信号名	方向	描述
addr[11:2]	I	下条指令地址
dout[31:0]	O	下条指令

3) 功能定义

序号	功能名称	功能描述
1	读取指令并输出	根据输入地址读取指令并输出

4、dm模块定义

1) 基本描述

DM的主要功能是数据的存储和读出。

2) 模块接口

信号名	方向	描述
addr [11:2]	I	传入的数据地址
we	I	是否允许写入数据 1: 允许写入 0: 不允许写入
clk	I	时钟信号
din	I	要存储的数据
dout	O	根据地址取出的数据

3) 功能定义

序号	功能名称	功能描述
1	读数据	根据读入的地址寻找相应地址的数据并读出，主要用于lw指令
2	写数据	sw指令将读出的寄存器的值存入dm

5、regfile模块定义

1) 基本描述

regfile的主要功能是完成寄存器的读取写数据。

2) 模块接口

信号名	方向	描述
readRegister1 [4:0]	I	指令的[25:21]位, rs段
readRegister2 [4:0]	I	指令的[20:16]位, rt段
writeRegister [4:0]	I	指明往哪个寄存器写数据, 由指令的类型决定
writeData	I	指明往寄存器中写入的数据是什么
regWrite	I	决定是否写数据 1: 写 0: 不写
clk	I	时钟信号
rst	I	复位信号 1: 复位, 寄存器清零 0: 无效
readData1 [31:0]	O	readRegister1对应的寄存器中的数据
readData2 [31:0]	O	readRegister2对应的寄存器中的数据

3) 功能定义

序号	功能名称	功能描述
1	读寄存器	根据指令读取指定寄存器
2	读数据	根据指令从指定寄存器中取数据
3	写数据	根据指令将数据写入到指定寄存器

6、alu模块定义

1) 基本描述

ALU的主要功能是完成数据的运算。ALU可以计算加法、减法、或、与、异或等运算。ALU除了完成相关运算, 还可判断beq指令下zero是否为0; slt指令下目的寄存器该存入1还是0。

2) 模块接口

信号名	方向	描述
aluOp[2:0]	I	控制alu执行何种运算 3'b000: 加 3'b001: 减 3'b010: 或 3'b011: slt 3'b100: 与 3'b101: 异或
data1 [31:0]	I	输入数据一，为rs寄存器中数据
data2 [31:0]	I	输入数据二，根据指令类型取寄存器数据或立即数
zero	O	判断beq指令下两数相减是否为0 1: 相等 0: 不相等
result [31:0]	O	数据运算结果

3) 功能定义

序号	功能名称	功能描述
1	数据运算	根据输入数值和aluOp进行运算
2	判断数据是否相等	用于输出zero
3	比较大小	用于在slt等指令下判断大小

7、mux模块定义

1) 基本描述

mux是多路选择器模块，根据选择信号从多个输入信号中选择一个作为输出信号。

2) 模块接口

mux2_32

信号名	方向	描述
a[31:0]	I	第一个32位数据
b[31:0]	I	第二个32位数据
sel	I	选择信号
y[31:0]	O	输出32位数据

mux2_32

信号名	方向	描述
a[14:0]	I	第一个15位数据
b[14:0]	I	第二个15位数据
sel	I	选择信号
y[14:0]	O	输出15位数据

mux2_5

信号名	方向	描述
a[4:0]	I	第一个5位数据
b[4:0]	I	第二个5位数据
sel	I	选择信号
y[4:0]	O	输出5位数据

mux2_30

信号名	方向	描述
a[31:2]	I	第一个30位数据
b[31:2]	I	第二个30位数据
sel	I	选择信号
y[31:2]	O	输出30位数据

mux3_32

信号名	方向	描述
a[31:0]	I	第一个32位数据
b[31:0]	I	第二个32位数据
c[31:0]	I	第三个32位数据
sel	I	选择信号
y[31:0]	O	输出32位数据

mux3_5

信号名	方向	描述
a[4:0]	I	第一个4位数据
b[4:0]	I	第二个4位数据
c[4:0]	I	第三个4位数据
sel	I	选择信号
y[4:0]	O	输出4位数据

3) 功能定义

序号	功能名称	功能描述
1	选择数据	根据选择信号从多个输入中选择一个数据输出

8、sl2模块定义

1) 基本描述

将输入的32位数据左移两位。虽然可以设置成自定义位数，感觉在这里有点鸡肋。

2) 模块接口

信号名	方向	描述
data[31:0]	I	输入的要移位的数据
slData[31:0]	O	移位后的数据

3) 功能定义

序号	功能名称	功能描述
1	移位	将数据左移两位

9、signext模块定义

1) 基本描述

extender的主要功能是扩展16位的立即数。其内部包括0扩展、符号扩展、高位扩展(lui)等扩展。

2) 模块接口

信号名	方向	描述
instruction[15:0]	I	输入的16位拓展数据
extType[1:0]	I	判断拓展的类型 2'b00: 符号拓展 2'b01: 无符号拓展 2'b10: lui拓展
signExtNumber[31:0]	O	将数据拓展后的结果

3) 功能定义

序号	功能名称	功能描述
1	无符号拓展	无符号拓展
2	有符号拓展	将数的最高位往前复制16个
3	高位拓展	在数后添加16个0

10、ctrl模块定义

1) 基本描述

Controller的主要功能是控制各个部件的执行功能及多路选择器。Controller是根据指令决定各个控制信号的。

2) 模块接口

信号名	方向	描述
opcode [5:0]	I	指令的[31:26]位
funct [5:0]	I	指令的[5:0]位
regDst [1:0]	O	决定写入到哪个寄存器 2'b00: 写入到指令[20:16]指定的寄存器 2'b01: 写入到指令[15:11]指定的寄存器 2'b10: 写入到31号寄存器
jump [1:0]	O	跳转信号 2'b00: 无跳转 2'b01: j 2'b10: jr 2'b11: jal
branch [1:0]	O	分支信号 2'b00: 无分支 2'b10: beq 2'b11: bne
memRead	O	是否允许从dm中读数据，默认可读 1: 读 0: 不读
memToReg [1:0]	O	写入寄存器数据来源 2'b00: alu的计算结果写入 2'b01: dm读取的数据写入 2'b10: 时钟信号写入
aluOp [2:0]	O	控制alu执行何种运算 3'b000: 加 3'b001: 减 3'b010: 或 3'b011: slt 3'b100: 与 3'b101: 异或
memWrite	O	是否允许往dm中写入数据 1: 允许 0: 不允许
aluSrc	O	决定alu的第二个数据来源 1: alu的第二个数据来自extender扩展后的数 0: alu的第二个数据来自读出的第二个寄存器的值
regWrite	O	是否往寄存器中写入数据 1: 是 0: 否
extType [1:0]	O	判断拓展的类型 2'b00: 符号拓展 2'b01: 无符号拓展 2'b10: lui拓展

3) 功能定义

序号	功能名称	功能描述
1	控制	根据指令输出各种数据控制各个部件执行

11、datapath模块定义

1) 基本描述

Datapath是以上各个部件的综合，将他们之间的相互连线都连接在一起。

2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	复位信号 1: 复位 0: 无效
regDst [1:0]	I	决定写入到哪个寄存器 2'b00: 写入到指令[20:16]指定的寄存器 2'b01: 写入到指令[15:11]指定的寄存器 2'b10: 写入到31号寄存器
jump [1:0]	I	跳转信号 2'b00: 无跳转 2'b01: j 2'b10: jr 2'b11: jal
branch [1:0]	I	分支信号 2'b00: 无分支 2'b10: beq 2'b11: bne
memRead	I	是否允许从dm中读数据，默认可读 1: 读 0: 不读
memToReg [1:0]	I	写入寄存器数据来源 2'b00: alu的计算结果写入 2'b01: dm读取的数据写入 2'b10: 时钟信号写入
aluOp [2:0]	I	控制alu执行何种运算 3'b000: 加 3'b001: 减 3'b010: 或 3'b011: slt 3'b100: 与 3'b101: 异或
memWrite	I	是否允许往dm中写入数据 1: 允许 0: 不允许
aluSrc	I	决定alu的第二个数据来源 1: alu的第二个数据来自extender扩展后的数 0: alu的第二个数据来自读出的第二个寄存器的值
regWrite	I	是否往寄存器中写入数据 1: 是 0: 否

信号名	方向	描述
extType [1:0]	I	判断拓展的类型 2'b00: 符号拓展 2'b01: 无符号拓展 2'b10: lui拓展
instruction[31:0]	O	当前执行的指令

3) 功能定义

序号	功能名称	功能描述
1	数据传送	实现数据连接

12、mips模块定义

1) 基本描述

mips是datapath和controller的组合，可以通过clk和rst来控制mips指令的执行。

2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	复位信号 1: 复位 0: 无效

3) 功能定义

序号	功能名称	功能描述
1	执行指令	执行相应的指令

13、add模块定义

1) 基本描述

与门，用来判断branch和zero指令是不是同时成立从而诱发beq/bne。

2) 模块接口

信号名	方向	描述
branch	I	分支信号
zero	I	zero信号
PCSrc	O	跳转信号 1: 跳转 0: 不跳转

3) 功能定义

序号	功能名称	功能描述
1	与门	输入数据与运算

14、compare模块定义

1) 基本描述

判断输入数据是否相等，服务于beq/bne指令。

2) 模块接口

信号名	方向	描述
compare_data1[31:0]	I	比较数1
compare_data2[31:0]	I	比较数2
zero	O	相等信号 1：相等 0：不相等

3) 功能定义

序号	功能名称	功能描述
1	比较器	比较输入数据相等与否

15、forward_unit_branch模块定义

1) 基本描述

解决beq指令的数据冒险

2) 模块接口

信号名	方向	描述
IF_ID_Rs[4:0]	I	IF_ID的Rs寄存器序号
IF_ID_Rt[4:0]	I	IF_ID的Rt寄存器序号
EX_MEM_Rd[4:0]	I	EX_MEM的目标寄存器序号
EX_MEM_regWrite	I	EX_MEM的写寄存器信号
ID_EX_Rd[4:0]	I	ID_EX的目标寄存器标号
ID_EX_regWrite	I	ID_EX的写寄存器信号
Forward_Rs[1:0]	O	转发信号 2'b00: 寄存器数据 2'b01: alu转发 2'b10: MEM转发
Forward_Rt[1:0]	O	转发信号 2'b00: 寄存器数据 2'b01: alu转发 2'b10: MEM转发

3) 功能定义

序号	功能名称	功能描述
1	转发信号输出	计算转发信号

16、forward_unit_alu模块定义

1) 基本描述

解决在alu的数据的转发问题，处理alu部件的数据冒险

2) 模块接口

信号名	方向	描述
ID_EX_Rs[4:0]	I	ID_EX的Rs寄存器序号
ID_EX_Rt[4:0]	I	ID_EX的Rt寄存器序号
EX_MEM_Rd[4:0]	I	EX_MEM的目标寄存器序号
EX_MEM_regWrite	I	EX_MEM的写寄存器信号
MEM_WB_Rd[4:0]	I	MEM_WB的目标寄存器标号
MEM_WB_regWrite	I	MEM_WB的写寄存器信号
Forward_A[1:0]	O	转发信号 2'b00: 寄存器数据 2'b01: MEM转发 2'b10: WB转发
Forward_A[1:0]	O	转发信号 2'b00: 寄存器数据 2'b01: MEM转发 2'b10: WB转发

3) 功能定义

序号	功能名称	功能描述
1	转发信号输出	计算转发信号

17、hazard_detection模块定义

1) 基本描述

冒险检测模块，通过阻塞解决lw-use、lw-branch冒险

2) 模块接口

信号名	方向	描述
jump[1:0]	I	是否跳转
ID_EX_Rt[4:0]	I	ID_EX的Rt寄存器序号
IF_ID_Rs[4:0]	I	IF_ID的Rs寄存器序号
IF_ID_Rt[4:0]	I	IF_ID的Rt寄存器信号
ID_EX_memRead	I	ID_EX的读dm信号
EX_MEM_Rt[4:0]	I	MEM_WB的目标寄存器信号
PCSrc	I	跳转信号 1: 跳转 0: 不跳转
EX_MEM_memRead	I	MEM_WB的读寄存器信号
PC_write	O	控制PC是否改变 1: 改变 0: 不改变
IF_ID_Write	O	控制IF_ID是否改变 1: 改变 0: 不改变
stall_info	O	阻塞信号 1: 阻塞 0: 不阻塞
flush	O	flush信号 1: flush 0: 不flush

3) 功能定义

序号	功能名称	功能描述
1	冒险检测处理	检测指定的一些冒险并进行解决

18、if_id模块定义

1) 基本描述

if_id流水寄存器

2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	清零信号
fourPc[31:0]	I	当前指令的PC值+4
instruction[31:0]	I	传来的指令
IF_ID_write	I	hazard detection传来的写信号
out_fourPC	O	将输入的fourPC保存并输出
out_instruction	O	将instruction保存并输出

3) 功能定义

序号	功能名称	功能描述
1	冒险检测处理	携带if阶段的数据往前走

19、id_ex模块定义

1) 基本描述

id_ex流水寄存器。

2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	清零信号
fourPC[31:0]	I	当前指令对应pc
regDst[1:0]	I	寄存器选择
memRead	I	dm读指令
memToReg[1:0]	I	寄存器写入数据选择信号
aluOp[2:0]	I	alu指令
memWrite	I	dm写指令
aluSrc	I	alu输入数据选择指令
regWrite	I	寄存器写指令
readData1[31:0]	I	寄存器读出数据1
readData2[31:0]	I	寄存器读出数据2
instruction1[4:0]	I	Rs指令
instruction2[4:0]	I	Rt指令
extNumber[31:0]	I	指令后16位拓展后的结果
instruction[31:0]	I	指令
out_fourPC[31:2]	O	保存并输出当前指令PC
out_regDst[1:0]	O	保存并输出寄存器选择信号
out_memRead	O	保存并输出dm读指令
out_memToReg[1:0]	O	保存并输出寄存器写入数据选择信号
out_aluOp[2:0]	O	保存并输出alu信号
out_memWrite	O	保存并输出dm写信号
out_aluSrc	O	保存并输出alu输入数据选择信号
out_regWrite	O	保存并输出寄存器写信号
out_readData1[4:0]	O	保存并输出寄存器读取数据1
out_readData2[4:0]	O	保存并输出寄存器读取数据2
out_instruction1[4:0]	O	保存并输出Rs
out_instruction2[4:0]	O	保存并输出Rt
out_extNumber[31:0]	O	保存并输出立即数拓展的结果
out_instruction[31:0]	O	保存并输出当前指令

3) 功能定义

序号	功能名称	功能描述
1	数据保存与传递	保存上级传来数据并传出，实现流水

20、ex_mem模块定义

1) 基本描述

ex_mem流水寄存器。

2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	清零信号
fourPC[31:0]	I	指令对应PC
memRead	I	dm读信号
memToReg[1:0]	I	写入寄存器数据选择信号
memWrite	I	dm写信号
regWrite	I	寄存器写信号
zero	I	零信号
aluResult[31:0]	I	alu计算结果
readData2[31:0]	I	将要写入寄存器的数据
writeDataReg[4:0]	I	写入寄存器地址
instruction[31:0]	I	当前指令
out_fourPC[31:0]	O	保存并输出指令对应PC
out_memRead	O	保存并输出dm读信号
out_memToReg[1:0]	O	保存并输出写入寄存器数据选择信号
out_memWrite	O	保存并输出dm写信号
out_regWrite	O	保存并输出寄存器写信号
out_zero	O	保存并输出零信号
out_aluResult[31:0]	O	保存并输出alu计算结果
out_readData2[31:0]	O	保存并输出将要写入寄存器的数据
out_writeDataReg[4:0]	O	保存并输出写入寄存器地址
out_instruction[31:0]	O	保存并输出当前指令

3) 功能定义

序号	功能名称	功能描述
1	数据保存与传递	保存上级传来数据并传出，实现流水

21、mem_wb模块定义

1) 基本描述

mem_wb流水寄存器。

2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	清零信号
fourPC[31:0]	I	指令对应PC
memToReg[1:0]	I	写入寄存器数据选择信号
aluResult[31:0]	I	alu计算结果
readData[31:0]	I	要写入dm的数据
writeDataReg[4:0]	I	写入寄存器的序号
instruction[31:0]	I	指令
regWrite	I	寄存器写信号
out_fourPC[31:0]	O	保存并输出指令对应PC
out_memToReg[1:0]	O	保存并输出写入寄存器数据选择信号
out_aluResult[31:0]	O	保存并输出alu计算结果
out_readData[31:0]	O	保存并输出要写入dm的数据
out_writeDataReg[4:0]	O	保存并输出写入寄存器的序号
out_instruction[31:0]	O	保存并输出指令
out_regWrite	O	保存并输出寄存器写信号

3) 功能定义

序号	功能名称	功能描述
1	数据保存与传递	保存上级传来数据并传出，实现流水

四、测试代码及结果

测试代码1


```
while:
addiu $t1,$zero,10
addiu $t2,$zero,0x0
addiu $t3,$zero,10
addiu $t4,$zero,1
addiu $t7,$zero,4
addiu $v0,$zero,0
```

```
for:
slt $v1,$v0,$t3
beq $v1,0,after1
addi $v0,$v0,1
sw $t1,($t2)
sub $t1,$t1,$t4
addi $t2,$t2,4
j for
```

```
after1:
addiu $t1,$zero,10
addiu $v0,$zero,0
addiu $a0,$zero,0
sub $t3,$t3,$t4
```

```
bubble1:
addiu $t2,$zero,0x0
slt $v1,$v0,$t3
beq $v1,0,after2
addi $v0,$v0,1
addiu $a0,$zero,0
```

```
bubble2:
slt $v1,$a0,$t3
beq $v1,0,bubble1
addi $a0,$a0,1
lw $t1,($t2)
lw $t5,4($t2)
addi $t2,$t2,4
slt $v1,$t1,$t5
beq $v1,1,bubble2
sub $t2,$t2,$t7
ori $t6,$t1,0
and $t1,$t1,$zero
or $t1,$t1,$t5
and $t5,$t5,$zero
or $t5,$t5,$t6
sw $t1,($t2)
sw $t5,4($t2)
add $t2,$t2,$t7
```

```

j bubble2

after2:
addiu $v0,$zero,0
addiu $t2,$zero,0x0

result:
slt $v1,$v0,$s3
beq $v1,0,afterall
addi $v0,$v0,4
lw $t1 ($t2)
addi $t2,$t2,4
j result

afterall:
lui $t1,0x0100
xor $t1,$t1,$t1
addiu $t1,$zero,10
j end

end:
j end

```

对应二进制

```


























2409000a
240a0000
240b000a
240c0001
240f0004
24020000
004b182a
20010000
10230005
20420001
ad490000
012c4822
214a0004
08000c06
2409000a
24020000
24040000
016c5822
240a0000
004b182a
20010000
10230016
20420001
24040000
008b182a
20010000

```

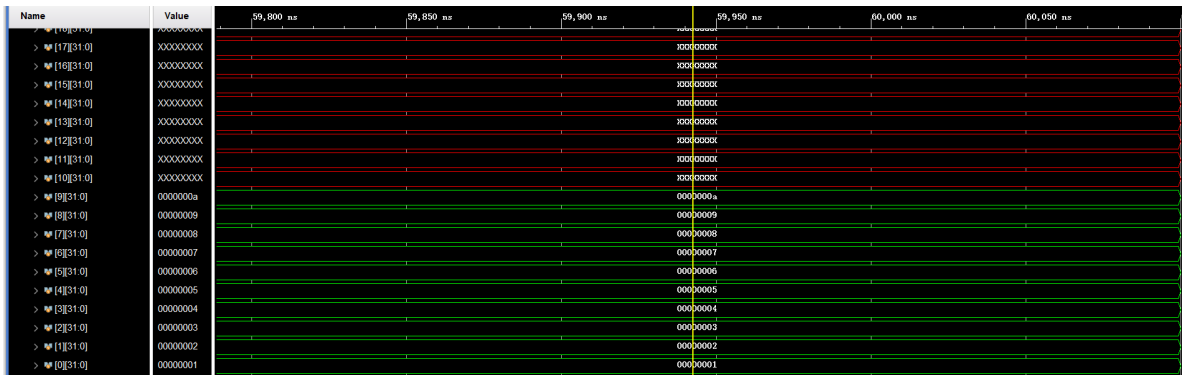
1023fff7
20840001
8d490000
8d4d0004
214a0004
012d182a
20010001
1023fff6
014f5022
352e0000
01204824
012d4825
01a06824
01ae6825
ad490000
ad4d0004
014f5020
08000c18
24020000
240a0000
0053182a
20010000
10230004
20420004
8d490000
214a0004
08000c2e
3c090100
01294826
2409000a
08000c39
08000c39

运行结果

register

>  [24]31.0]	00000000	00000000
>  [23]31.0]	00000000	00000000
>  [22]31.0]	00000000	00000000
>  [21]31.0]	00000000	00000000
>  [20]31.0]	00000000	00000000
>  [19]31.0]	00000000	00000000
>  [18]31.0]	00000000	00000000
>  [17]31.0]	00000000	00000000
>  [16]31.0]	00000000	00000000
>  [15]31.0]	00000004	00000004
>  [14]31.0]	00000002	00000002
>  [13]31.0]	0000000a	0000000a
>  [12]31.0]	00000001	00000001
>  [11]31.0]	00000009	00000009
>  [10]31.0]	00000000	00000000
>  [9]31.0]	0000000a	0000000a
>  [8]31.0]	00000000	00000000
>  [7]31.0]	00000000	00000000
>  [6]31.0]	00000000	00000000
>  [5]31.0]	00000000	00000000
>  [4]31.0]	00000009	00000009
>  [3]31.0]	00000000	00000000
>  [2]31.0]	00000000	00000000
>  [1]31.0]	00000000	00000000
>  [0]31.0]	00000000	00000000

dm



测试代码2

```
addiu $t1,$zero,10
addiu $t2,$zero,0x0
addiu $t3,$zero,8
sw $t1,($t2)
lw $t3,($t2)
beq $t1,$t3,after1
addiu $t1,$zero,5

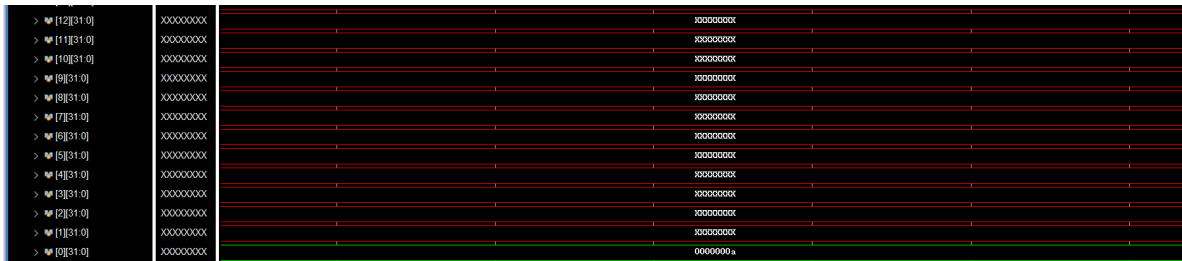
after1:
addiu $t1,$zero,10
```

对应二进制

```
2409000a
240a0000
240b0008
ad490000
8d4b0000
112b0001
24090005
2409000a
```

运行结果

dm



register


```
20020008
20030008
ac030000
00621820
8c030000
10430003
20030009
2004000a
2005000b
2008000c
2009000d
```

运行结果

dm

> [14][31:0]	XXXXXXXX	XXXXXXXXXX
> [11][31:0]	XXXXXXXX	XXXXXXXXXX
> [10][31:0]	XXXXXXXX	XXXXXXXXXX
> [9][31:0]	XXXXXXXX	XXXXXXXXXX
> [8][31:0]	XXXXXXXX	XXXXXXXXXX
> [7][31:0]	XXXXXXXX	XXXXXXXXXX
> [6][31:0]	XXXXXXXX	XXXXXXXXXX
> [5][31:0]	XXXXXXXX	XXXXXXXXXX
> [4][31:0]	XXXXXXXX	XXXXXXXXXX
> [3][31:0]	XXXXXXXX	XXXXXXXXXX
> [2][31:0]	XXXXXXXX	XXXXXXXXXX
> [1][31:0]	XXXXXXXX	XXXXXXXXXX
> [0][31:0]	00000008	00000008

register

> [24][31:0]	00000000	00000000
> [23][31:0]	00000000	00000000
> [22][31:0]	00000000	00000000
> [21][31:0]	00000000	00000000
> [20][31:0]	00000000	00000000
> [19][31:0]	00000000	00000000
> [18][31:0]	00000000	00000000
> [17][31:0]	00000000	00000000
> [16][31:0]	00000000	00000000
> [15][31:0]	00000000	00000000
> [14][31:0]	00000000	00000000
> [13][31:0]	00000000	00000000
> [12][31:0]	00000000	00000000
> [11][31:0]	00000000	00000000
> [10][31:0]	00000000	00000000
> [9][31:0]	00000000	00000004
> [8][31:0]	00000000	0000000c
> [7][31:0]	00000000	00000000
> [6][31:0]	00000000	00000000
> [5][31:0]	00000000	00000000
> [4][31:0]	00000000	00000000
> [3][31:0]	00000008	00000008
> [2][31:0]	00000008	00000008
> [1][31:0]	00000000	00000000
> [0][31:0]	00000000	00000000

五、实验时间安排

第一次完成流水是大概花了六小时，实现了流水作业。

五一的时候大概花了十几个小时实现了当时上课所讲的内容，即控制冒险之前；

本周末从周六下午开始做，周六先理了一下思路，然后开始了新加模块，但是晚上调试进度缓慢。

周日从上午九点多开始调试，发现了一些问题，大概在下午一点完成，并给助教验收，然后又进行了一些小修改，删去了一些无用的传输数据，同时将mem转发去掉，改成了双阻塞。

总时长大约40小时。

六、心得体会

- 1、流水线会出现时空不对应的情况，就需要好好的思考，并且习惯这样的调试方法。
- 2、在写之前，需要好好的设计一下，有哪些冒险，冒险的类型，是一级冒险还是二级冒险，如何去解决。因为在这里的冒险是一环扣一环的，我们不能穷举所有冒险的可能，所以我们就需要对冒险先分类，并且给出对应的解决方案，这样才不至于写的太乱。同时，一个冒险可能执行以后会划归到另一个冒险，所以我们如果一开始设计好了，他就自然而然的接上了，不需要临时再去考虑。
- 3、注意一些很小的错误，比如数位错了（寄存器五位但是写成[5:0]），接错了数据等情况，这些都是很难找出来但是如果一开始就避免会很好的错误。
- 4、注意初始值的设定。
- 5、流水能提高效率，但同时带来的时空不同步导致的问题，才需要我们去用hazard detection模块和forward模块去解决。也就是说，我们加入这些部件，是看到了问题去解决问题而去添加的，而且我觉得是很符合人的思维的，比如某个数据取不回来，那我就让他取回来就可以。但是并不是都可以这样操作，有时候考虑到一些因素，我们会选择用一种看似不好的方式来处理，相当于曲线救国的感觉，并不是一条直线。
- 6、github很有用，可以出现问题及时的回滚。