计算机组成原理实验报告参考模板

一、CPU设计方案综述

(一）总体设计概述

本CPU为Verilog实现的流水线MIPS-COU,支持的指令集包含{addu,subu,ori, lw, sw, beq, lui, nop,sll, j, jr, jal, addiu,jalr,slti,bgez}。CPU主要包含了PC,NPC,IM,GRF,ALU,DM,EXT,CTRL,.FWandSCTRL,MUX等模块，这些模块按照数据通路、控制信号,冒险控制的设计逐级展开。

（二）关键模块定义

1. GRF

信号名	方向	描述
clk	I	CPU时钟
reset	I	同步复位信号
WE	I	写使能
A1[4:0]	I	rs输入
A2[4:0]	I	rt输入
A3[4:0]	I	rd输入
WD[31:0]	I	寄存器文件写入数据
PC[31:0]	I	current pc
RD1[31:0]	O	读寄存器文件时寄存器rs的输出
RD2[31:0]	O	读寄存器文件时寄存器rt的输出

2. NPC

信号名	方向	描述
PC[31:0]	I	当前PC值
IMM26[25:0]	I	j型指令的26位立即数
IMM16[15:0]	I	b型指令的16位立即数
IMM32[31:0]	I	jr型指令的寄存器数据
NPCOP[2:0]	I	npc控制指令
Zero[4:0]	I	第0位beq信号，1为跳转，0为pc+4；第一位bgez信号，1为跳转，0为pc+4
NPC[31:0]	O	下一个PC的值
PCA4[31:0]	O	当前PC+4

功能定义

OP值	功能
000	pc+4
001	B类指令
010	JAL类指令，跳26Imm
011	JR类指令，跳寄存器

3. ALU

信号名	方向	描述
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
SHAMT[4:0]	I	移位信号
ALUOP[3:0]	I	ALU功能控制
ALUOUT[31:0]	O	ALU计算结果

功能定义

OP值	功能
0000	A+B
0001	A-B
0010	A|B
0011	A&B
0100(lui)	B<<16
0101(SLL)	B<< shamt
0110(SLTI)	若A小于B，则输出1，否则0

CMP

信号名	方向	描述
CMPOP[3:0]	I	控制信号
Rs[31:0]	I	转发后的Rs
Rt[31:0]	I	转发后的Rt
Zero	O	1为真0为假，对于当前B指令

```
BEQ 4'b0000
BGEZ 4'b0001
```

4.CTRL

信号名	方向	描述
InstrD[31:0]	I	D级指令
NPCOPD[2:0]	O	npc功能控制
ExtopInD[1:0]	O	D级扩展控制
DmweToM	O	数据文件写使能,流水到M级
RFWE	O	grf写使能流水到W级
A3[4:0]	O	根据指令选择的A3
RFWDMUX[2:0]	O	grf写入控制
ALUBMUX[2:0]	O	ALUB端控制
ALUOP[3:0]	O	ALUOP,流水到E
DMOP[2:0]	O	流水到M级控制DM写入
TuseRs[2:0]	O	Rs的Tuse
TuseRt[2:0]	O	Rt的Tuse
TnewE[2:0]	O	指令在E级的Tnew
CMPOP[3:0]	O	CMP控制信号

信号控制矩阵

指令	指令类型	指令码	NPCOP	RFWE	ALUOP	EXTOP	DMWE	A3	RFWDMUX	ALUBMUX	DMOP	CMPOP
ADDU	R	100001	PC+4(000)	1	ADD(0000)	XX	0	InstrD[15:11]	000	000	000	XXXX
SUBU	R	100011	PC+4	1	SUB(0001)	XX	0	InstrD[15:11]	000	000	000	XXXX
ORI	I	001101	PC+4	1	ori(0010)	01	0	InstrD[20:16]	000	001	000	XXXX
LW	I	100011	PC+4	1	ADD	00	0	InstrD[20:16]	001	001	000	XXXX
SW	I	101011	PC+4	0	ADD	00	1	XXX	XXX	001	000	XXXX
BEQ	I	000100	B(001)	0	SUB	XX	0	XXX	XXX	000	000	0000
LUI	I	001111	PC+4	1	LUI	01	0	InstrD[20:16]	000	001	000	XXXX}
SLL	R	000000	PC+4	1	SLL(0101)	XX	0	InstrD[15:11]	000	000	000	XXXX
J	J	000010	010(J)	0	XXXX	XX	0	XXX	XXX	XXX	000	XXXX
JAL	J	000011	010	1	XXXX	XX	0	31	010	XXX	000	XXXX
JR	R	001000	011(JR)	0	XXXX	XX	0	XXX	XXX	XXX	000	XXXX
ADDIU	I	001001	PC+4	1	ADD	00	0	InstrD[20:16]	000	001	000	XXXX
BGEZ	I	000001	001	0	XXXX	00	0	XXX	XXX	XXX	000	0001
JALR	R	001001	011	1	XXXX	XX	0	InstrD[15:11]	010	XXX	000	XXXX
SLTI	I	001010	PC+4	1	0110	00	0	InstrD[20:16]	000	001	000	XXXX
LB	I	100000	PC+4	1	ADD	00	0	InstrD[20:16]	011	001	000	XXXX
SB	I	101000	PC+4	0	ADD	00	1	XXX	XXX	001	001	XXXX
ADD(无溢出)	R	100000	PC+4	1	ADD	XX	0	InstrD[15:11]	000	000	XXX	XXXX
ADDI(无溢出)	I	001000	PC+4	1	ADD	00	0	InstrD[20:16]	000	001	XXX	XXXX

5.FWandSCTRL

信号名	方向	描述
A1D[4:0]	I	
A2D[4:0]	I	
A1E[4:0]	I	
A2E[4:0]	I	
A1M[4:0]	I	
A2M[4:0]	I	
A3M[4:0]	I	
A3W[4:0]	I	
WEE	I	
WEM	I	
WEW	I	
TuseRs[2:0]	I	
TuseRt[2:0]	I	
TnewE[2:0]	I	
TnewM[2:0]	I	
FWCMPRS[2:0]	O	cmpRS转发
FWCMPRT[2:0]	O	cmpRT转发
FWALURS[2:0]	O	ALURS转发
FWALURT[2:0]	O	ALURT转发
FWDMRT[2:0]	O	DM写入数据转发
Stall	O	停止信号

T译码	指令
TuseRs	
2	
1	ADDU SUBU ORI LW SW LUI SLTI ADDIU ADD ADDI
0	BEQ JR JALR
TuseRt	
2	SW
1	ADDU SUBU SLL ADD
0	BEQ
TnewE	
2	LW
1	ADDU SUBU ORI LUI SLL SLTI ADDIU ADD ADDI
0	JAL JALR

5.MUX

RFWDMUX

信号名	方向	描述
RFWDOP[2:0]	I	控制信号
ALUOUT[31:0]	I	ALU计算结果
DMOUT[31:0]	I	数据存储器的输出数据
PCA4[31:0]	O	PC+4
RFWD[31:0]	O	写入GRF的数据

选择	功能
000	ALUOUT
001	DMOUT
010	PC+8
011	DMOUT[7:0]符号扩展

ALUBMUX

信号名	方向	描述
ALUBOP[2:0]	I	控制信号
rt[31:0]	I	转发后RT寄存器的值
IMM16[31:0]	I	经过EXT扩展后的16位立即数
ALUB[31:0]	O	ALUB的输入

选择	功能
000	rt
001	imm16(I型指令)

6. DM

信号名	方向	描述
DMOP[2:0]	I	写入数据方式选择
Address[31:0]	I	数据存储地址
Input[31:0]	I	输入数据
clk	I	全局时钟
Reset	I	同步清零
DMWE	I	写使能
PC[31:0]	I	pc
Data[31:0]	O	输出数据

选择	功能
000	32imm
001	sb8位

7. EXT

信号名	方向	描述
EXTOP[2:0]	I	选择信号
imm16[15:0]	I	16位立即数
shamt[4:0]	I	5位立即数
extout[31:0]	O	输出

选择	功能
00	符号扩展16-32
01	无符号扩展16-32

8. IFU(分为PC和IM)

- PC

信号名	方向	描述
npc[31:0]	I	下个PC值
reset	I	同步复位至0x0000_3000
clk	I	时钟
pc[31:0]	O	当前PC

- IM

信号名	方向	描述
Addr[23:0]	I	当前指令地址
instr[31:0]	O	当前指令

9.流水线寄存器

```
module PipeD (
    input  clk,
    input reset,
    input stall,
    input [31:0] InstrInF,
    input [31:0] PCA4F,
    input [31:0] PCF,
    output reg [31:0] PCD,
    output reg [31:0] PCA4D,
    output reg [31:0] InstrInD
);
```

```
module PipeE (
    input clk,
    input reset,
    input [31:0] PCD,
    input [31:0] InstrD,
    input RFWEInD,
    input DMWEInD,
    input [4:0] A3InD,
    input [31:0] IMM16EXTD,
    input [2:0] TnewEinD,
    input [2:0] RFWDMUXIND,
    input [2:0] ALUBMUXIND,
    input [3:0] ALUOPIND,
    input [1:0] EXTOPIND,
```

```
    input [2:0] DMOPIND,
    input [31:0] RsInD,
    input [31:0] RtInD,
    output reg [31:0] PCE,
    output reg [31:0] InstrE,
    output reg RFWEInE,
    output reg DMWEInE,
    output reg [4:0] A3InE,
    output reg [31:0] IMM16EXTE,
    output reg [2:0] TnewE,
    output reg [2:0] RFWDMUXINE,
    output reg [2:0] ALUBMUXINE,
    output reg [3:0] ALUOPINE,
    output reg [1:0] EXTOPINE,
    output reg [2:0] DMOPINE,
    output reg [31:0] RsInE,
    output reg [31:0] RtInE
);
```

```
module PipeM (
    input clk,
    input reset,
    input [31:0] PCE,
    input [31:0] InstrE,
    input RFWEInE,
    input DMWEInE,
    input [4:0] A3InE,
    input [2:0] TnewE,
    input [2:0] RFWDMUXINE,
    input [2:0] DMOPINE,
    input [31:0] RtFromE,
    input [31:0] ALUOUTINE,
    output reg [31:0] PCM,
    output reg [31:0] InstrM,
    output reg RFWEInM,
    output reg DMWEInM,
    output reg [4:0] A3InM,
    output reg [2:0] TnewM,
    output reg [2:0] RFWDMUXINM,
    output reg [2:0] DMOPINM,
    output reg [31:0] RtInM,
    output reg [31:0] ALUOUTINM
);
```

```
module PipeW (
    input clk,
    input reset,
    input [31:0] PCM,
    input [31:0] InstrM,
    input RFWEInM,
    input [4:0] A3InM,
    input [2:0] RFWDMUXINM,
```

```
    input [31:0] ALUOUTINM,
    input [31:0] DMOUTM,
    output reg [31:0] PCW,
    output reg [31:0] InstrW,
    output reg RFWEInW,
    output reg [4:0] A3InW,
    output reg [2:0] RFWDMUXINW,
    output reg [31:0] ALUOUTINW,
    output reg [31:0] DMOUTW
);
```

10.转发Mux

```
module  FWCMPRS(input [2:0] FWOP,
                input [31:0] DataFromM,
                input [31:0] DataFromW,
                input [31:0] DataFromD,
                output [31:0] RsToCmp
);
```

```
module FWCMPRT(input [2:0] FWOP,
                input [31:0] DataFromM,
                input [31:0] DataFromW,
                input [31:0] DataFromD,
                output [31:0] RtToCmp
);
```

```
module FWALURS (input [2:0] FWOP,
                input [31:0] DataFromM,
                input [31:0] DataFromW,
                input [31:0] DataFromE,
                output [31:0] RsToAlu

);
```

```
module FWALURT(input [2:0] FWOP,
                input [31:0] DataFromM,
                input [31:0] DataFromW,
                input [31:0] DataFromE,
                output [31:0] RtToAlu

);
```

```
module FWDRT (input [2:0] FWOP,
              input [31:0] DataFromM,
              input [31:0] DataFromW,
              output [31:0] WDToDM

);
```

（三）重要机制实现方法

1. 跳转

NPC模块在D级传递给CMP和NPC控制信号，CMP控制B类指令的Zero信号，CMPOP控制CMP判断的是哪条B指令的Zero,Zero信号传递到NPC中协同B指令跳转。NPC信号选择是什么类型的跳转，类型有J(跳26imm),JR(跳转发后的Rs,无转发视为原地转发)，B(跳imm16，zero判断是否跳),和F级PC+4.

2. 流水线延迟槽

跳转指令在D级时，PC+4的指令在F级，跳转指令只更新PC值，PC+4指令进入D级，该指令即为延迟槽。同时JAL应用PC+8写入$31

3. 转发和暂停

在D级的CTRL模块对D级指令进行AT译码，将TuseRs和TuseRt和A3D送入转发暂停单元.

```
module FWandSCTRL(
    input [4:0] A1D,
    input [4:0] A2D,
    input [4:0] A1E,
    input [4:0] A2E,
    input [4:0] A1M,
    input [4:0] A2M,
    input [4:0] A3E,
    input [4:0] A3M,
    input [4:0] A3W,
    input WEE,
    input WEM,
    input WEW,
    input [2:0] TuseRs,
    input [2:0] TuseRt,
    input [2:0] TnewE,
    input [2:0] TnewM,
    output [2:0] FWCMPRS,
    output [2:0] FWCMPRT,
    output [2:0] FWALURS,
    output [2:0] FWALURT,
    output [2:0] FWDMRT,
    output Stall
    );  // FW
    //FWCMP
    assign FWCMPRS = (A1D==A3M && WEM && A3M)?`CMPFROMM:
                     (A1D==A3W && WEW && A3W)?`CMPFROMW:`CMPFROMD;

    assign FWCMPRT = (A2D==A3M && WEM && A3M)?`CMPFROMM:
                     (A2D==A3W && WEW && A3W)?`CMPFROMW:`CMPFROMD;
```

```
    //FWALURS
    assign FWALURS = (A1E==A3M && WEM && A3M)?`ALUFROMM:
                     (A1E==A3W && WEW && A3W)?`ALUFROMW:`ALUFROME;

    assign FWALURT = (A2E==A3M && WEM && A3M)?`ALUFROMM:
                     (A2E==A3W && WEW && A3W)?`ALUFROMW:`ALUFROME;

    //FWDMRT
    assign FWDMRT = (A2M==A3W && WEW && A3W)?1:0;


    //Stall
    wire StallRsE,StallRsM,StallRtE,StallRtM;
    assign StallRsE = (TuseRs<TnewE) && (A1D) && (A1D==A3E) &&WEE;
    assign StallRsM = (TuseRs<TnewM) && (A1D) && (A1D==A3M) && WEM;
    assign StallRtE = (TuseRt<TnewE) && (A2D) && (A2D==A3E) && WEE;
    assign StallRtM = (TuseRt<TnewM) && (A2D) && (A2D==A3M) && WEM;
    assign Stall = StallRsE|StallRsM|StallRtE|StallRtM;

endmodule
```

根据DEMW级的A1A2A3和WE信号转发，通过Tuse和Tnew控制暂停

二、测试方案

（一）典型测试样例

```
ori $28, 0
ori $29, 0
#data
li $8, 1
li $9, 11
li $10, 111
li $11, 1111
li $12, 11111
li $13, 111111
li $14, 1111111
li $15, 11111111
##addr
ori $16, 0
ori $17, 20
ori $18, 40
ori $19, 60
ori $20, 80
ori $21, 100
sw $8, 0($16)
sw $9, 4($16)
sw $10, 8($16)
sw $11, 12($16)
sw $12, 16($16)
sw $13, 20($16)
sw $14, 24($16)
sw $15, 28($16)
##zhuanfa
## 1
lw $1, 0($16)
```

```
sw $1, 8($17)
nop
## 2
lw $2, 4($16)
addu $10, $10, $11
sw $2, 0($18)
# 3
addu $1, $3, $4
subu $2, $4, $3
ori $3, 151
sw $1, 16($16)
#stall
#4
ori $4, 4
sw $4, 0($0)
nop
nop
nop
##
lw $17, 0($0)
sw $4, 0($17)
##
#5
addu $7, $8, $9
##
addu $6, $8, $9
beq $6, $7, branch1
##
subu $3, $14, $15
nop
ori $25, 56
branch1:
#6
jal branch2
addu $31, $31, $31
#
nop
branch2:
subu $31, $31, $26
ori $31, 0x30e8
nop
nop
nop
nop
##
lw $4, 0($0)
beq $6, $4, branch2
##
nop
lw $4, 0($0)
ori $1, 0
beq $6, $4, branch2
```

（二）自动测试工具

讨论区

三、思考题

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

答:数据通路:PC的值根据NPC的值更新

控制信号:D级CTRL单元控制NPC行为选择，分为PC+4,B类指令，Jimm26指令，J寄存器指令三种。指令进入D级是如果是分支指令，则PC的值根据分支指令更新，否则PC+4

2. 对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8？

答:跳转指令后有延迟槽，故写入寄存器的正确地址应该是延迟槽指令下一条指令，即为PC+8。

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

答:将数据传递到需求者有一定时间延迟，如果由ALU或者DM来提供数据，则会提高该级一个周期的时间，从而有可能使流水线一个周期的时间提高，降低了流水线的效率，背离的流水线的设计初衷。而用流水线寄存器提供数据，则不会提高周期时间。

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

```
lw $3, 0($1)
addu $8, $8, $8
sw $3, 4($2)
```

如果不采用转发过的数据，则sw在M级时lw更新了$3的数据无法转发到M级中，导致错误。解决办法就是sw在M级采用E级被转发过的$3(此时lw在W级，转发$3到E级)。

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

因为可能D级读取的寄存器在W级正要被写入。不采用内部转发可以采用外部转发，将W级的数据通过转发mux转发到D级。

Thinking 3：为什么 0 号寄存器需要特殊处理？

一条指令前面的指令可能会对0号寄存器写入指令，如果不特殊处理，对0号寄存器写入的数据可能被错误转发

Thinking 4：什么是“最新产生的数据”？

对当前指令寄存器的值有影响的最低级次的流水线寄存器产生的写入寄存器的数据

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

在AT法翻译A3时，如果WE为0，则将A3改为0，即与“供给者需求者的 A 相同，且不为 0”的行为相同，不必判断we

在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的?

1.

运算类 **$1**, $x, $y

b类 **$1**, $x, im

2.

load类 **$1**, x($y)

b类 **$1**, $x, im

3.

load类 **$1**, x($y)

XXXXX

b类 **$1**, $x, im

4.

load类 **$1**, x($y)

运算类 $x, **$1**, $y

5.

运算类 **$1**, $x, $y

jr **$1**

6.

load类 **$1**, x($y)

jr **$1**

7.

load类 **$1**, x($y)

XXXXX

jr **$1**

8.

load类 **$1**, x($y)

load类 $x, y(**$1**)

9.

load类 **$1**, x($y)

store类 $x, x(**$1**)

上述需要暂停解决

1.

load类 **$1**, $x, $y

store类 **$1**, x($y)

上述W级向M级转发

2.

load类 **$1**, x($y)

XXXXX

store类 **$1**, x($y)

上述流水线寄存器同步转发信息

3.

运算类 **$1**, $x, $y

XXXXX

XXXXX

store类 **$1**, x($y)

上述GRF内部转发

测试用例见上