计算机组成原理实验报告(P6)

一、 CPU设计方案综述

（一）、总体设计概述

本CPU为Verilog实现的五级流水线MIPS - CPU，支持的指令集：

MIPS-C3={LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、 SUBU、 MULT、MULTU、 DIV、 DIVU、 SLL、 SRL、 SRA、 SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO}，可以支持除浮点运算外的绝大多数定点类程序的运行。

为了实现这些功能CPU的设计分为三级， 第一级为流水级，包含（F级、F\D流水寄存器、D级、D\E流水寄存器、E级、E\M流水寄存器、M级、M\W流水寄存器），第二级为各流水级的内部模块，第三级为各模块的内部构造。具体的层次结构如图所示：

对于各级指令，采用分布式译码的方式，在每一级中设置独立但完全相同的Controller, 并通过采取阻塞转发机制，处理冲突和各类冒险，在保证指令执行正确性的前提下，提高效率。最后，通过自己编写的对针对各种转发机制的测试程序，验证其正确性。

指令数据通路的分类

R1	**add、addu、sub、subu、or、nor、xor、and、slt、sltu、sllv、srav、srlv**
ls	**lb、lbu、lh、lhu、lw、sb、sh、sw**
R2	**sll、sra、srl**
R3	**jr、jalr**
RI	**addi、addiu、andi、lui、ori、slti、sltiu、xori**
B	**beq、bgez、bgtz、blez、bltz、bne**
J	**j、jal**
MD	**div、divu、mult、multu**
DC	**mfhi、mflo、mthi、mtlo**

在P5的基础上进行增量开发，增加或增扩如下信号：

选择信号	ALUOp	指令
add	0000	add、addu、addi、addiu
sub	0001	sub、subu
and	0010	and、andi
or	0011	or、ori、lui
nor	0100	nor
xor	0101	xor、xori
sll	0110	sll、sllv
sra	0111	sra、srav
srl	1000	srl、srlv
slt	1001	slt、slti
sltu	1010	sltu、sltiu

BSet	选择信号	指令
R[rs] = R[rt]	000	beq
R[rs]! = R[rt]	001	bne
R[rs]>0	010	bgtz
R[rs]>=0	011	bgez
R[rs]<0	100	bltz
R[rs]<=0	101	blez

WDSel	选择信号	指令
ALUResult	00	add、addu、sub、subu、or、nor、xor、and、slt、sltu、sllv、srav、srlv、sll、sra、srl、addi、addiu、andi、lui、ori、slti、sltiu、xori、mflo、mfhi
Memresult	01	lb、lbu、lh、lhu、lw
PC+8	10	jal、jalr

WRSel	选择信号	指令
rt	00	addi、addiu、andi、lui、ori、slti、sltiu、xori、lb、lbu、lh、lhu、lw
rd	01	add、addu、sub、subu、or、nor、xor、and、slt、sltu、sllv、srav、srlv、sll、sra、srl、mflo、mfhi
$31	10	jal、jalr

RFWr	选择信号	指令
不写	0	others
写寄存器	1	add、addu、sub、subu、or、nor、xor、and、slt、sltu、sllv、srav、srlv、sll、sra、srl、addi、addiu、andi、lui、ori、slti、sltiu、xori、lb、lbu、lh、lhu、lw、mflo、mfhi

DFWr	选择信号	指令
不写	0	others
写DM	1	sb、sh、sw

ExtOp	选择信号	指令
signed	00	addiu、addi、slti、sltiu
unsigned	01	xori、ori、andi
lui	10	lui

ALUSrc	选择信号	指令
rt	0	others
Extresult	1	addi、addiu、andi、lui、ori、slti、sltiu、xori

NPCOp	选择信号	指令
PC+4	00	others
B	01	beq、bgez、bgtz、blez、bltz、bne 且Zero信号为1
j	10	j、jal
jr	11	jr、jalr

Mem_type	选择信号	指令
不读取信号	00	others
byte	01	sb、lb、lbu
half	10	sh、lh、lhu
word	11	sw、lw

乘除法及相关指令

REOp	选择信号	指令
读取hi\lo寄存器	01\10	mfhi\mflo
ALU计算结果	00	others
选择PC+8	11	jal、jalr

HiLoWr	选择信号	指令
写入Hi寄存器	01	mthi
写入Lo寄存器	10	mtlo
不写入寄存器	00	others

Start	选择信号	指令
MD工作启动	1	mult multu div divu
MD不工作	0	others

名称	I\O	端口大小	说明
reset	I	1	同步复位信号
clk	I	1	时钟信号
RD1	I	31:0	当前指令指针
RD2	I	31:0	下一个指令指针
start	I	1	指令寄存器使能
HiLoWr	I	1:0	HiLo寄存器写使能
stall	O	1	阻塞信号
Hi	O	31:0	hi寄存器的值
Lo	O	31:0	lo寄存器的值

在冒险控制中，增加MDStall信号（stall|start）当IF处的指令为mfhi、mflo、mthi、mtlo、mult、multu、div、divu 时，阻塞这里的指令。

（二）、关键模块定义

1.F级

1.PC

名称	I\O	端口大小	说明
PC	O	31:0	当前指令指针
NPC	I	31:0	下一个指令指针
clk	I	1	时钟信号
reset	I	1	同步复位信号
PCEnD	I	1	指令寄存器使能

2. NPC

名称	I\O	端口大小	说明
PC	I	31:0	当前指令指针
NPCOp	I	1:0	00:PC+4, 01:beq, 10:jal, 11:jr
IMM	I	25:0	jal 写入地址
RA	I	31:0	jr 跳转地址
PC4	O	31:0	PC+4
NPC	O	31:0	下一个指令指针

3.IM

名称	I\O	端口大小	说明
Iaddr	I	31:0	pc指针
Instr	O	31:0	当前指令

2.F\D寄存器

名称	I\O	端口大小	说明
clk	I	1	时钟信号
reset	I	1	同步复位信号
FDEn_FD_I	I	1	寄存器使能端
Instr_FD_I	I	31:0	FD指令输入（F级当前指令）
PC_FD_I	I	31:0	FDpc输入（F级当前pc值）
PC_FD_O	O	31:0	FDpc输出（D级当前pc值）
Instr_FD_O	O	31:0	FD指令输出（D级当前指令）

3. D级

1.GRF

名称	I\O	端口大小	说明
A1	I	4:0	rs地址（25:21）
A2	I	4:0	rt地址（20:16）
A3	I	4:0	WR选择信号 00：rs（20:16），01：rd（15:10），$ra （5'h1f）
WD	I	31:0	写入信号
RD1	O	31:0	rs寄存器的值
RD2	O	31:0	rt寄存器的值

2. Extender

名称	I\O	端口大小	说明
ExtOp	I	1:0	00: signed_extend 01: unsigned_extend 10: lui({in[15:0]}, {16{0}})
in	I	15:0	Instr[15:0]
out	O	32:0	输出结果

3.Acoder

对各个指令的AT信息进行译码

名称	I\O	端口大小	说明
Opcode	I	5:0	Opcode选择信号
Funct	I	5:0	Funct选择信号
Tusers	O	2:0	rs的使用时间
Tusert	O	2:0	rt的时钟时间
Tnew	O	2:0	产生结果的时间

4.D\E寄存器

名称	I\O	端口大小	说明
clk	I	1	时钟信号
reset	I	1	同步复位信号
Instr_DE_I	I	31:0	FD指令输入（D级当前指令）
PC_DE_I	I	31:0	FDpc输入（D级当前pc值）
PC_DE_O	O	31:0	FDpc输出（E级当前pc值）
Instr_DE_O	O	31:0	FD指令输出（E级当前指令）
RD1_DE_I	I	31:0	value of rs(D级)
RD2_DE_I	I	31:0	value of rt(D级)
Ext_DE_I	I	31:0	Ext resut(D级)
RD1_DE_I	O	31:0	value of rs(E级)
RD2_DE_I	O	31:0	value of rt(E级)
Ext_DE_O	O	31:0	Ext resut(E级)
Tnew_DE_I	I	2:0	冒险控制Tnew(D级)
Tnew_DE_O	O	2:0	冒险控制Tnew(E级)

5. E级

1.ALU

名称	I\O	端口大小	说明
SrcA	I	31:0	参与运算的A信号
SrcB	I	31:0	参与运算的B信号
Zero	O	31:0	beq相等信号
result	O	31:0	计算结果
ALUControl	I	31:0	ALU控制信号

2.WD_MUX

对写入的目标寄存器进行选择 2'b00选择rt寄存器， 2'b01选择rd寄存器 2'b10选择$31.

3. ALU_SRC_MUX

对ALU的第二个输入信号进行选择，0时选择rt寄存器的值作为输入，1时选择Extender的结果作为输入.

6.E\M寄存器

名称	I\O	端口大小	说明
clk	I	1	时钟信号
reset	I	1	同步复位信号
Instr_EM_I	I	31:0	EM指令输入（E级当前指令）
PC_EM_I	I	31:0	EMpc输入（E级当前pc值）
PC_EM_O	O	31:0	EMpc输出（M级当前pc值）
Instr_EM_O	O	31:0	EM指令输出（M级当前指令）
ALURS_EM_I	I	31:0	ALU计算结果(E级)
WD_EM_I	I	31:0	WD结果(E级)
Dst_EM_I	I	31:0	可能写入的寄存器地址(E级)
ALURS_EM_O	O	31:0	ALU计算结果(M级)
WD_EM_O	O	31:0	WD结果(M级)
Dst_EM_O	O	31:0	可能写入的寄存器地址(M级)
Tnew_EM_I	I	2:0	冒险控制Tnew(E级)
Tnew_EM_O	O	2:0	冒险控制Tnew(M级)

7.M级

1. DM

名称	I\O	端口大小	说明
scr	I	1	写数据使能
sel	I	1	RAM工作使能
A	I	4:0	写入地址
D1	I	31:0	写入的数据
D2	O	31:0	输出的数据
clk	I	1	时钟信号
reset	I	1	同步复位信号

8.M\W寄存器

名称	I\O	端口大小	说明
clk	I	1	时钟信号
reset	I	1	同步复位信号
Instr_MW_I	I	31:0	MW指令输入（E级当前指令）
PC_MW_I	I	31:0	MWpc输入（E级当前pc值）
PC_MW_O	O	31:0	MWpc输出（M级当前pc值）
Instr_MW_O	O	31:0	MW指令输出（M级当前指令）
ALURS_MW_I	I	31:0	ALU计算结果(M级)
RD_MW_I	I	31:0	WD结果(M级)
Dst_MW_I	I	31:0	可能写入的寄存器地址(M级)
ALURS_MW_O	O	31:0	ALU计算结果(W级)
RD_MW_O	O	31:0	WD结果(W级)
Dst_MW_O	O	31:0	可能写入的寄存器地址(W级)
Tnew_MW_I	I	2:0	冒险控制Tnew(M级)
Tnew_MW_O	O	2:0	冒险控制Tnew(W级)

9.W级

1.LTC

名称	I\O	端口大小	说明
addr	I	31:0	读取的地址（根据最后两位判断指令）
M_type	I	1:0	Load选择信号
Din	I	31:0	原始数据（1w）
Dout	I	31:0	选择后的数据（lw, lh, lb）

2. WDMUX

对写回GRF的数据进行选择 2'b00选择由ALU直接产生的结果， 2'b01选择从DM中读取的数据，2'b01写入PC值（PC+8）.

10. Controller

对于每一级中均独立存在的Controller,其控制信号的组合逻辑如下

名称	I\O	端口大小	说明
Opcode	I	5:0	opcode字段
Func	I	4:0	function字段

控制信号及对应的指令如下：

Instru	Opcode	RFWr	WRSel	WDSel	ALU_SrcSel	DMWr	ALUOp	NPCOp	ExtOp	M_type
R_type（or and addu subu sll sllv)	000000	1	01	00	0	0	ALUControl	00	x	00
ori	001101	1	00	00	1	0	011	00	01	00
lw	100011	1	00	01	1	0	000	00	00	11
lh	100001	1	00	01	1	0	000	00	00	10
lb	100000	1	00	01	1	0	000	00	00	01
sw	101011	0	x	00	1	1	000	00	00	11
sh	101001	0	x	00	1	1	000	00	00	10
sb	101000	0	x	00	1	1	000	00	00	01
beq	000100	0	x	00	0	0	001	01	x	00
lui	001111	1	00	00	1	0	000	00	10	00
addiu	001001	1	00	00	1	0	000	00	00	00
jal	000011	1	10	10	x	0	000	10	x	00
jr	000000	0	x	11	x	0	000	11	x	00
j	000010	0	x	00	x	x	000	10	x	00

（二）冒险控制

在本次CPU设计中，由于哈佛体系下DM,IM的分离，不存在结构冒险，对于跳转冒险，由于仅包含beq这一由寄存器的值决定是否发生跳转的指令，故将跳转指令的冒险归入数据冒险之中。

使用D级中的ACoder部件，各指令AT信息译码：

指令	使用寄存器编号	写入寄存器编号	Tuse	Tnew
addu\subu\or\and\sll\sllv	rs,rt	rd	1	2
load(lw,lh,lb)	rs	rt	1	3
sw(sw,sh,sb)	rs,rt	/	(rs)1,(rt)2	0
lui	/	rt	/	2
ori	/	rt	/	2
beq	rs,rt	/	0	/
jr	rs	/	0	/
jal	/	$31	/	3
addiu	rs	rt	1	2

对于AT信息的控制

通过RiskCtrl部件对冒险进行控制

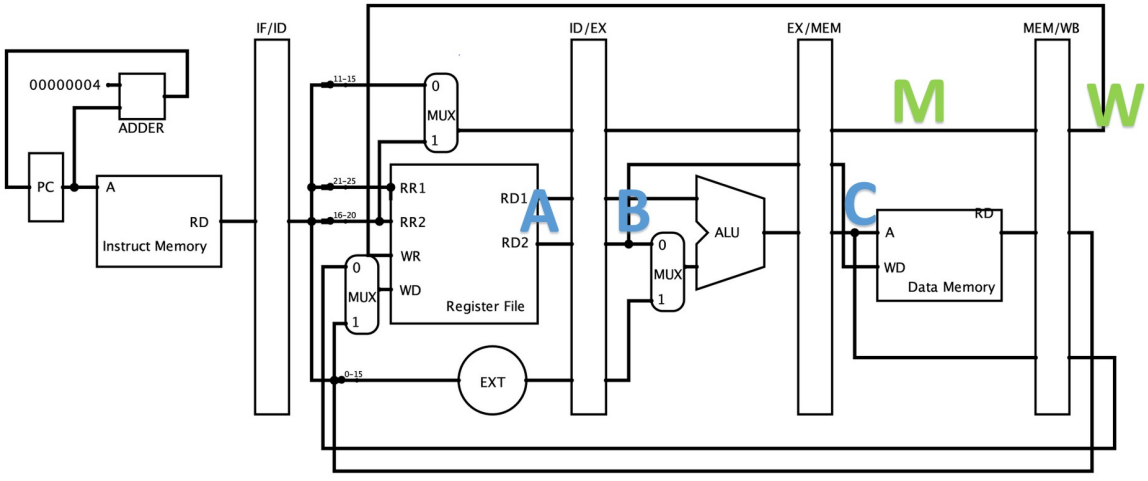
指令	使用寄存器编号	写入寄存器编号	Tusers	Tusert	Tnew
addu、add、sub、subu、or、and、nor、xor、sllv、srav、srlv、slt、sltu	rs,rt	rd	1	1	2
sll、sra、srl	rs	rd	5	1	2
load(lw、lh、lb、lhu、lbu)	rs	rt	1	5	3
sw(sw、sh、sb)	rs,rt	/	1	2	0
ori、addiu、addi、andi、xori、slti、sltiu	rs	rt	1	5	2
lui	/	rt	5	5	2
beq、bgez、bgtz、blez、bltz、bne	rs,rt	/	0	0	0
jr	rs	/	0	0	0
jal	/	$31	5	5	2
jalr	rs	rd	0	5	2
j	/	/	5	5	0
mthi、mtlo	rs	hi/lo	1	0	0
mflo、mfhi	/	rd	5	5	2

二、测试方案

1.正确性检验

设计的测试程序方案如下：

首先对不含mult、multu、div、divu、mfhi、mflo、mthi、mtlo的指令集进行测试，测试覆盖的方法类似P5，只需在在每一类测试中增加指令。



如图，本次设计的CPU除去乘除相关的指令后有三个接受转发点，记为A,B,C点，有两个提供转发数据点，记作M,W点，故可以产生AM,AW,BM,BW,CW共五种路径的转发（不存在CW路径），继而考虑每条路径上有两条路，则共有10条路需要进行单独测试，此外，还需考虑多路转发条件均成立时选择的优先级是否正确，因此需要进一步考虑 AM,AW同时成立的数据转发，BM,BW同时成立的数据转发，在每一种类别中充分测试各个指令以及$zero的特判），即可对转发和暂停的冒险控制的正确性进行强度较高的测试。

1、数据类指令测试

```
lui $t3, 10
sw $t3, 0($0)
sw $t3, 4($0)

#BM转发测试_rs
ori $t1, $t1, 1
addu $t1, $t1, $t2
nop
addi $t3, $3, 2
add $t4, $t3, $t2
nop
addi $t4, $t4, 2
sub $t5, $t4, $t6
nop
sllv $t5, $t5, $t1
sra $t6, $t5, 1
nop
srav $t7, $t7, $t3
sltiu $t5, $t7, 123
nop
xori $t6, $t5, 13
nor $t7, $t6, $t5
nop
addiu $t2, $t2, 1
subu $t2, $t2, $t3
nop
lui $t1,  1
addu $0, $t2, $1
subu $t2, $0, $t3
addu $t1, $t1, $t2
nop
subu $t1, $t2, $t2
subu $t1, $t1, $t2
nop
subu $t1, $t1, $t1
ori $t1, $t1, 1
nop
subu $t2, $t2, 0
subu $t2, $t2, $t1
nop
subu $t3, $t2, $t1
subu $t2, $t3, $t1
nop
addu $t3, $t2, $t3
addu $t3, $t3, $t2
nop
lw $t2, 4($0)
addu $t3, $t2, $t1
```

```
nop

#BM转发测试_rt
ori $t1, $t1, 1
addu $t1, $t2, $t1
nop
addi $t3, $3, 2
add $t4, $t2, $t3
nop
addi $t4, $t4, 2
sub $t5, $t6, $t4
nop
sllv $t5, $t5, $t1
sra $t6, $t5, 1
nop
srav $t7, $t7, $t3
sltiu $t5, $t7, 123
nop
xori $t6, $t5, 13
nor $t7, $t5, $t6
nop
addiu $t2, $t2, 1
subu $t2, $t3, $t2
nop
lui $t1,  1
addu $0, $t2, $1
subu $t2, $0, $t3
addu $t1, $t1, $t2
nop
subu $t1, $t2, $t2
subu $t1, $t1, $t2
nop
subu $t1, $t1, $t1
ori $t1, $t1, 1
nop
subu $t2, $t2, 0
subu $t2, $t2, $t1
nop
subu $t3, $t2, $t1
subu $t2, $t3, $t1
nop
addu $t3, $t2, $t3
addu $t3, $t3, $t2
nop
lw $t2, 4($0)
addu $t3, $t2, $t1
nop

#BW转发测试_rs
ori $t1, $t1, 1
nop
addu $t1, $t1, $t2
nop
addi $t3, $3, 2
nop
add $t4, $t3, $t2
nop
addi $t4, $t4, 2
```

```
nop
sub $t5, $t4, $t6
nop
sllv $t5, $t5, $t1
nop
sra $t6, $t5, 1
nop
srav $t7, $t7, $t3
nop
sltiu $t5, $t7, 123
nop
xori $t6, $t5, 13
nop
nor $t7, $t6, $t5
nop
addiu $t2, $t2, 1
nop
subu $t2, $t2, $t3
nop
addiu $t1, $t1, 1
nop
lui $t1,  45
nop
lw $t3, 0($0)
nop
addiu $t3, $t3, 2
nop
ori $t1, $t1, 1
nop
addu $t1, $t1, $t1
nop
subu $t1, $t1, $t3
nop
and $t1, $t1, $t2
nop
addu $t3, $t1, $t2
nop
addu $0, $t3, $t2
nop
subu $t5, $0, $1
nop

#BW转发测试_rt
ori $t1, $t1, 1
nop
addu $t1, $t2, $t1
nop
addi $t3, $3, 2
nop
add $t4, $t2, $t3
nop
addi $t4, $t4, 2
nop
sub $t5, $t6, $t4
nop
sllv $t5, $t5, $t1
nop
sra $t6, $t5, 1
```

```
nop
srav $t7, $t7, $t3
nop
sltiu $t5, $t7, 123
nop
xori $t6, $t5, 13
nop
nor $t7, $t5, $t6
nop
addiu $t2, $t2, 1
nop
addiu $t1, $t1, 1
nop
lui $t1,  45
nop
lw $t3, 0($0)
nop
addiu $t3, $t3, 2
ori $t1, $t1, 1
nop
addu $t1, $t1, $t1
nop
subu $t1, $t3, $t1
nop
and $t3, $t2, $1
nop
addu $t3, $t1, $t3
nop
subu $0, $t2, $t5
nop
addu $t2, $t3, $t0
nop
nop

#BW与BM转发的优先顺序测试_rs
andi $t3, $t3, 0
andi $t2, $t2, 0
ori $t3, 1
ori $t2, 2
srav $t2, $t2, $t3
slti $t3, $t3, 3
or $t2, $t2, $t3
and $t2, $t3, $t2
sltu $t4, $t2, $t3
slt $t2, $t4, $t3
add $t3, $t2, $t3
sub $t5, $t3, $t2
xor $t6, $t3, $t2
nor $t7, $t6, $t8
addu $t2, $t7, $t2
subu $t4, $t2, $t3
addiu $t4, $t4, -1
addu $t4, $t4, $t2
or $t1, $t4, $t2
subu $t5, $t1, $t2
nop
nop
```

```
#BW与BM转发的优先顺序测试_rt
andi $t3, $t3, 0
andi $t2, $t2, 0
ori $t3, 1
ori $t2, 2
srav $t2,$t3, $t2
slti $t3, $t3, 3
or $t2, $t2, $t3
and $t2, $t3, $t2
sltu $t4, $t3, $t2
slt $t2, $t3, $t4
add $t3, $t3, $t2
sub $t5, $t3, $t2
xor $t6, $t3, $t2
nor $t7, $t6, $t6
ori $t3, 4
ori $t2, 5
addu $t2, $t2, $t2
subu $t4, $t3, $t2
addiu $t4, $t4, -1
addu $t4, $t4, $t2
addiu $t4, $t4, -1
addu $t4, $t3, $t4
or $t1, $t5, $t4
subu $t5, $t2, $t1
nop
nop

# CW指令测试
ori $t2, $t2, 3
sw $t2, 16($0)
lw $t3, 16($0)
sw $t3, 20($0)
and $t2, $t2, $0
addiu $t2, $t2, 4
sw $t2, 0($0)
lw $t2, 0($0)
sw $t3, 0($t2)
ori $t3, $t3, 4
sw $t3, 4($0)
lw $0, 0($t2)
sw $0, 4($0)
```

2、B类指令正确性测试：

```
#B_type(AM)测试
andi $t6, $t6, 0
andi $t7, $t7, 0
ori $t6, $t6, 1
ori $t7, $t7, 1
beq $t6, $t7, label_beq
nop
addi $t1, $t1, 1
label_beq:
```

```
addu $t1, $t1, $t1
beq $t1, $t1, nobe_beq
nop
addu $t1, $t1, $t1
nop
nobe_beq:
nop
nop

andi $t6, $t6, 0
andi $t7, $t7, 0
ori $t6, $t6, 1
ori $t7, $t7, 1
blt $t6, $t7, label_blt
nop
addi $t1, $t1, 1
label_blt:
addu $t1, $t1, $t1
beq $t1, $t1, nobe_blt
nop
addu $t1, $t1, $t1
nop
nobe_blt:

andi $t6, $t6, 0
andi $t7, $t7, 0
ori $t6, $t6, 1
ori $t7, $t7, 1
bltz $t6, label_bltz
nop
addi $t1, $t1, 1
label_bltz:
sub $t1, $t1, $t1
addi $t1, $t1, -1
bltz $t1, nobe_bltz
nop
addu $t1, $t1, $t1
nop
nobe_bltz:

andi $t6, $t6, 0
andi $t7, $t7, 0
ori $t6, $t6, 1
ori $t7, $t7, 1
bgtz $t6, label_bgtz
nop
addi $t1, $t1, 1
label_bgtz:
sub $t1, $t1, $t1
addi $t1, $t1, -1
bgtz $t1, nobe_bgtz
nop
addu $t1, $t1, $t1
nop
nobe_bgtz:

andi $t6, $t6, 0
andi $t7, $t7, 0
```

```
ori $t6, $t6, 1
ori $t7, $t7, 1
bne $t6, $t7, label_bne
nop
addi $t1, $t1, 1
label_bne:
sub $t1, $t1, $t1
addi $t1, $t1, -1
bne $t1, $t1, nobe_bne
nop
addu $t1, $t1, $t1
nop
nobe_bne:

andi $t6, $t6, 0
andi $t7, $t7, 0
ori $t6, $t6, 1
ori $t7, $t7, 1
blez $t6, label_blez
nop
addi $t1, $t1, 1
label_blez:
sub $t1, $t1, $t1
addi $t1, $t1, -1
blez $t1, nobe_blez
nop
addu $t1, $t1, $t1
nop
nobe_blez:

andi $t6, $t6, 0
andi $t7, $t7, 0
ori $t6, $t6, 1
ori $t7, $t7, 1
bgez $t6, label_bgez
nop
addi $t1, $t1, 1
label_bgez:
sub $t1, $t1, $t1
addi $t1, $t1, -1
bgez $t1, nobe_bgez
nop
addi $t1, $t1, 1
nop
nobe_bgez:

#B_type(AW)测试
and $t6, $t6, $0
ori $t6, 4
sw $t6, 16($0)
lw $t1, 16($0)
ori $t7, $t7, 4
beq $t1, $t7 right2_beq
nop
addiu $t7, $t7, 2
right2_beq:
sw $t7, 0($0)
lw $t7, 4($0)
```

```
nop
nop

and $t6, $t6, $0
ori $t6, 4
sw $t6, 16($0)
lw $t1, 16($0)
ori $t7, $t7, 4
bne $t1, $t7 right2_bne
nop
addiu $t7, $t7, 2
right2_bne:
sw $t7, 0($0)
lw $t7, 4($0)
nop
nop

and $t6, $t6, $0
ori $t6, 4
sw $t6, 16($0)
lw $t1, 16($0)
ori $t7, $t7, 4
blt $t1, $t7 right2_blt
nop
addiu $t7, $t7, 2
right2_blt:
sw $t7, 0($0)
lw $t7, 4($0)
nop
nop

and $t6, $t6, $0
ori $t6, 4
sw $t6, 16($0)
lw $t1, 16($0)
ori $t7, $t7, 4
bgt $t1, $t7 right2_bgt
nop
addiu $t7, $t7, 2
right2_bgt:
sw $t7, 0($0)
lw $t7, 4($0)
nop
nop

and $t6, $t6, $0
ori $t6, 4
sw $t6, 16($0)
lw $t1, 16($0)
ori $t7, $t7, 4
bgtz $t1 right2_bgtz
nop
addiu $t7, $t7, 2
right2_bgtz:
sw $t7, 0($0)
lw $t7, 4($0)
nop
nop
```

```
and $t6, $t6, $0
ori $t6, 4
sw $t6, 16($0)
lw $t1, 16($0)
ori $t7, $t7, 4
bltz $t1 right2_bltz
nop
addiu $t7, $t7, 2
right2_bltz:
sw $t7, 0($0)
lw $t7, 4($0)
nop
nop

and $t6, $t6, $0
ori $t6, 4
sw $t6, 16($0)
lw $t1, 16($0)
ori $t7, $t7, 4
bgez $t1 right2_bgez
nop
addiu $t7, $t7, 2
right2_bgez:
sw $t7, 0($0)
lw $t7, 4($0)
nop
nop

#B_type(AM与AW的优先顺序测试)
and $t6, $t6, $0
and $t5, $t5, $0
ori $t5, 5
ori $t6, 6
addiu $t6, $t6, -1
beq $t6, $t5, right_seq1_beq
nop
addiu $t7, $t7, 2
right_seq1_beq:
sw $t7, 0($0)
ori $0, 4
beq $0, $0, right_seq13_beq
nop
addiu $t7 $t7, 1
right_seq13_beq:
nop
nop

and $t6, $t6, $0
and $t5, $t5, $0
ori $t5, 5
ori $t6, 6
addiu $t6, $t6, -1
ble $t6, $t5, right_seq1_ble
nop
addiu $t7, $t7, 2
right_seq1_ble:
sw $t7, 0($0)
```

```
ori $0, 4
ble $0, $0, right_seql3_ble
nop
addiu $t7 $t7, 1
right_seql3_ble:
nop
nop

and $t6, $t6, $0
and $t5, $t5, $0
ori $t5, 5
ori $t6, 6
addiu $t6, $t6, -1
blt $t6, $t5, right_seq1_blt
nop
addiu $t7, $t7, 2
right_seq1_blt:
sw $t7, 0($0)
ori $0, 4
blt $0, $0, right_seql3_blt
nop
addiu $t7 $t7, 1
right_seql3_blt:
nop
nop

and $t6, $t6, $0
and $t5, $t5, $0
ori $t5, 5
ori $t6, 6
addiu $t6, $t6, -1
bge $t6, $t5, right_seq1_bge
nop
addiu $t7, $t7, 2
right_seq1_bge:
sw $t7, 0($0)
ori $0, 4
bge $0, $0, right_seql3_bge
nop
addiu $t7 $t7, 1
right_seql3_bge:
nop
nop

and $t6, $t6, $0
and $t5, $t5, $0
ori $t5, 5
ori $t6, 6
addiu $t6, $t6, -1
blez $t6,  right_seq1_blez
nop
addiu $t7, $t7, 2
right_seq1_blez:
sw $t7, 0($0)
ori $0, 4
blez $0, right_seql3_blez
nop
addiu $t7 $t7, 1
```

```
right_seql3_blez:
nop
nop

and $t6, $t6, $0
and $t5, $t5, $0
ori $t5, 5
ori $t6, 6
addiu $t6, $t6, -1
bgez $t6,  right_seq1_bgez
nop
addiu $t7, $t7, 2
right_seq1_bgez:
sw $t7, 0($0)
ori $0, 4
bgez $0, right_seql3_bgez
nop
addiu $t7 $t7, 1
right_seql3_bgez:
nop
nop
```

3、J类指令测试

```
# 跳转指令及延迟槽测试
and $t1, $t1, $0
ori $t1, $t1, 100
jal funct
addiu $ra, $ra, 4
addiu $t1, $t1, 2
j end
nop
funct:
sw $t2, 0($0)
jr $ra
nop
end:
```

使用自制的测评辅助工具对结果进行校对（测评辅助工具代码详见附录），检验设计CPU转发控制的正确性。

4、乘除类指令测试

```
ori $t1, $t1, -1
ori $t2, $t2, -2
sw $t2, 0($0)
lw $t3, 0($0)
divu $t1, $t3
mfhi $s4
addu $s3, $s4, $t1
mflo $s3
ori $t6, $t6, 1
div $t1, $t2
```

```
mfhi $s4
mflo $s3
mult $t1, $t2
mfhi $s4
mflo $s3
lw $t1, 0($0)
multu $t1, $t2
ori $s2, $s2, 1
ori $t2, $t2, 1
andi $s4, $s4, 0
mfhi $s4
addu $s4, $s4, $s4
mflo $s3
addu $s3, $s3, $4
addi $t2, $t2, 1
mfhi $t7
lw $t3, 0($0)
mult $t3, $3
ori $t5, $t5, 1
mfhi $t4
mflo $t6
mthi $t5
mtlo $t6
mfhi $t7
mflo $t9
```

2.效率检验（正确的暂停机制）

在保证正确性的前提下，还需要考虑执行的效率，故设计了如下的测试程序，该程序设计时在认为设计正确的情况下不会出现暂停，若出现多余的暂停，则可以认为暂停的机制出现了问题，可以通过查看仿真波形的stall信号来确定，而前一步的正确性检验已经检验CPU的正确性，故而无需考虑不应该出现暂停的地方发生暂停的情况。

```
# 暂停机制测试（不含乘除类指令）
and $t2, $t2, $0
ori $t2, $t2, 2
lw $t1, 0($0)
nop
nop
beq $t1, $t2, jump
nop
ori $t3, $t3, 100
jump:
addu $t2, $t2, $t2
addu $t1, $t1, $t1
nop
beq $t1, $t2, jump2
nop
ori $t4, $t4, 100
jump2:
and $t1, $t1, $0
ori $t1, $t1, 100
jal funct  #延迟槽部分特判
addiu $ra, $ra, 4
addiu $t1, $t1, 2
```

```
j end
nop
funct:
sw $t2, 0($0)
jr $ra
nop
end:

# 暂停机制测试
divu $t1, $t3
mfhi $s4
addu $s3, $s4, $t1
mflo $s3
ori $t6, $t6, 1
mult $t1, $t2
mfhi $s4
addi $s4, $s4, 1
mflo $s3
mult $t1, $t2
mfhi $t4
mflo $t6
mthi $t5
mtlo $t6
mfhi $t7
mflo $t9
```

三、思考题

1.为什么需要有单独的乘除法部件而不是整合进ALU？为何需要有独立的HI、LO寄存器?

乘除法（尤其是除法）的计算速度较慢，如果整合进ALU，会导致ALU部件整体的延迟大大增加，会严重限制CPU主频的提高导致速度下降，因而设置独立的乘除部件与独立的HI、LO寄存器。可以在不执行乘除法指令时在一个周期内完成ALU的计算，而对于乘除法指令则设置多个周期进行计算（比如本次实验中模拟分别用5个、10个周期执行乘除法指令），同时，设立独立的HI、LO寄存器，可以保证调用乘除结果的独立性，即乘除法的计算结果只有通过mfhi、mflo指令存入GPR中才能被使用，这样增加了乘除法计算和其值调用的独立性，更有利于提高CPU的稳定性。

2.参照你对延迟槽的理解，试解释“乘除槽”。

在乘除法计算时（本实验中分别为5个周期和10个周期）若后续指令不是mflo、mfhi、mtlo、mthi，则可以继续执行（遇到这4条指令， 则必须暂停）这一部分与乘除法运算同时执行的指令所在的位置即可称之为类似延迟槽的”乘除槽“，在编写汇编程序或者进行编译设计时，可以像“延迟槽”一样利用“乘除槽”，以提高CPU的运行效率。

3.举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑C语言中字符串的情况）

C语言中，一个字符占据一个字节的空间，一个字符串占据一段连续的空间，在处理或打印字符串时，将会依次读取这段由字符串占据的值，当访问该字符串的第一个值时，需要从内存中读出，所用的时间较长。但之后该字符串由很大概率保存在cache中，能进行高速的访问。而若采取按字访问，则由于字符串所占空间的提升导致寻找数据时hit的概率下降，如出现miss，则需要再次以较慢的速度访问cache故此时，按字节访问内存更占据优势。类似地，当需要访问一段的连续的内存空间且一个单位的数据大小小于一个字时，按字节访问更占据优势。

4.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？

详见测试样例

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

1.控制信号与AT信息的取值问题——分类与相似

首先确定对指令的分类以及每类中P5已经实现的代表性指令

R1 （addu）	**add、addu、sub、subu、or、nor、xor、and、slt、sltu、sllv、srav、srlv**
ls（lw、sw）	**lb、lbu、lh、lhu、lw、sb、sh、sw**
R2（无）	**sll、sra、srl**
R3（jr）	**jr、jalr**
RI（ori）	**addi、addiu、andi、lui、ori、slti、sltiu、xori**
B（beq）	**beq、bgez、bgtz、blez、bltz、bne**
J（jal）	**j、jal**
MD	**div、divu、mult、multu**
DC	**mfhi、mflo、mthi、mtlo**

可以发现，每类中的指令与其类的代表指令的控制信号其实仅有一个不同，根据这些实际需要再进一步设计MUX对不同控制信号的选择，即可系统地解决数据通路的控制（不包含冲突控制）

类似地，对于AT信息，也用类似的方法获得最后的统计表如下。

指令	使用寄存器编号	写入寄存器编号	Tuse	Tnew
addu\subu\or\and\sll\sllv	rs,rt	rd	1	2
load(lw,lh,lb)	rs	rt	1	3
sw(sw,sh,sb)	rs,rt	/	(rs)1,(rt)2	0
lui	/	rt	/	2
ori	/	rt	/	2
beq	rs,rt	/	0	/
jr	rs	/	0	/
jal	/	$31	/	3
addiu	rs	rt	1	2

由于指令量较大，均采用控制信号驱动型的译码器进行译码。即可复杂性控制。

2.排列组合设计测试代码：由于种类类别较多，测试时很难保证全面性，故选择找到发出转发信号与接收转发信号的点，并对其进行排列组合（详见测试设计），在每一个组合中在确定可能的指令序列，可以在一定程度上提高测试的全面性。

3.宏定义：对于每条指令都定义属于它的一条宏，比如：

```
`define addiu  Opcode==6'b001001
`define sll    Opcode==6'b000000 && Funct==6'b000000
`define bgez   Opcode==6'b000001 && Special==5'b00001
```

这样定义的宏可以使得译码器的具体代码更加精简和直观，由便于今后的维护。

4.信号命名：对于需要跨模块的信号，均采用“端口名—I\O—所在模块名”的方式进行命名，且模块实例化时，“.接口(外部信号)”中使接口名与外部信号名相同，一定程度上虽然增大了代码量，但确实连线时的复杂性降低，正确率提升。

四、附录——辅助测试工具代码

主程序代码：

```
import subprocess
import os
from shutil import copyfile

# initialize path setting
path = "C:\\Users\\123\\Desktop\\computer organization\\P6\\test_tool4.0\\"
path0 = "C:\\Users\\123\\Desktop\\computer organization\\P6\\"

testfile = "test.asm"
# testfile = "MD1.asm"

# get code and standard output from mars(modified)
os.chdir(path)
print("1")
os.system("java -jar Mars.jar nc mc CompactDataAtZero dump .text HexText
code.txt > ans.txt " + testfile)
print("1")
copyfile((path+"code.txt"), (path+"mips\\code.txt"))
copyfile((path+"ans.txt"), (path+"report\\stdans.txt"))

# execute run.bat and get test output from I_verilog
os.chdir(path+"mips")
filepath = path+"mips\\run.bat"
p = subprocess.Popen(filepath, shell=True, stdout=subprocess.PIPE)
stdout, stderr = p.communicate()
copyfile(path+"mips\\myout.txt", path+"report\\myans.txt")

# compare two results and give feedback
os.chdir(path+"report")
std = open("stdans.txt", "r")
test = open("myans.txt", "r")
tlin_r = test.readline()
i = 0
while True:
    i = i + 1
    slin = std.readline()
    while slin[11:14] == "$ 0":
        slin = std.readline()
    tlin_r = test.readline()
    if len(slin) > 20 and slin[11] == '*':
        tem = list(slin)
        value = ord(tem[19]) - ord("0") if ord(tem[19]) - ord("0") < 10 else
ord(tem[19]) - ord("a") + 10
        value = value - value % 4
        value = value + ord("0") if value < 10 else value + ord("a") - 10
        tem[19] = chr(value)
        slin = "".join(tem)
    if len(tlin_r) > 20:
        tlin = tlin_r[20:-1] + tlin_r[-1]
    else:
        tlin = tlin_r
```

```
    if not tlin and slin == "\n":
        print("Accepted!")
        break
    elif not tlin:
        print("The answer is fewer than standard. We got nothing when we expect:
‘"+slin[0:-1]+"’ at line "+str(i))
        break
    elif not slin:
        print("The answer is too much. We got '"+tlin+"' when we expect
nothing")
        break
    elif slin != tlin:
        slin_next = std.readline()
        tlin_next_r = test.readline()
        if len(tlin_next_r) < 22:
            continue
        tlin_next = tlin_next_r[20:-1] + tlin_next_r[-1]
        if len(slin_next) > 20 and slin[11] == '*':
            tem = list(slin_next)
            value = ord(tem[19]) - ord("0") if ord(tem[19]) - ord("0") < 10 else
ord(tem[19]) - ord("a") + 10
            value = value - value % 4
            value = value + ord("0") if value < 10 else value + ord("a") - 10
            tem[19] = chr(value)
            slin_next = "".join(tem)
        if slin_next == tlin and tlin_next == slin:
            i = i + 1
            pass
        else:
            print("Wrong Answer! We got ‘"+tlin[0:-1]+"’ when we expect:
‘"+slin[0:-1]+"’ at line "+str(i))
            break
    else:
        pass

print("detailed information has been generated in report document")
a = input("do you want to copy instruction to ISim path(Y/N):")
if a == "y" or a == "Y":
    copyfile((path + "code.txt"), (path0 + "P6_1.0\\code.txt"))
    print("finished copy")
elif a == "N" or a == "n":
    pass
```

iverilog执行的批处理bat代码：

```
set iverilog_path=C:\iverilog\bin;
set gtkwave_path=C:\iverilog\gtkwave\bin;
set path=%iverilog_path%%gtkwave_path%%path%

iverilog -o "test_bench.vvp" "test_bench.v" "NPC.v" "PC.v" "Controller.v"
"ALU.v" "Extender.v" "LTC.v" "GRF.v" "IM.v" "mips.v" "DM.v" "ACoder.v" "D.v"
"F.v" "E.v" "M.v" "W.v" "EMReg.v" "DEReg.v" "MWReg.v" "RiskCtrl.v" "FDReg.v"

(vvp -n "test_bench.vvp")> myout.txt
```