CPU 设计文档

一、数据通路设计

1. FetchLevel.v

表 1 IFU 端口说明

信号名	方向	描述
CLK	I	时钟信号
RESET	I	复位信号 1:复位 0:无效
Stall_F	I	阻塞 PC 信号 1:阻塞 PC 0:无效
MUX_NPC_F_Sel [1:0]	I	选择进入 PC 的信号 00:PC4_F 01:PCBranch_D 10:RD1_D 11:EXT32_D
PC8_D [31:0]	I	D 级指令的 PC+8
PCBranch_D [31:0]		D 级指令计算得出的 PC 跳转位置
RD1_D [31:0]	I	D 级指令得出的 GRF 的 RD1
EXT32_D [31:0]	I	D 级指令经过 EXT 部件输出的结果
IR_F [31:0]	O	F 级指令
PC_F [31:0]	O	F 级 PC
PC4_F [31:0]	O	F 级 PC+4

表 2 IFU 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 被设置为 0x00003000
2	取指令	把指令从 IM 里拿出来
3	计算指令地址	计算下一条要执行的指令的地址。

2. RegisterFile.v

表 3 GRF 端口说明

信号名	方向	描述
CLK	I	时钟信号
RESET	I	复位信号 1:复位 0:无效
WE	I	写入信号 1:写操作 0:无效
PC[31:0]	I	W 级指令的 PC 值
A1[4:0]	I	读出数据至 RD1 的寄存器的地址
A2[4:0]	I	读出数据至 RD2 的寄存器的地址
A3[4:0]	I	要写入的寄存器的地址
WD[31:0]	I	要写入的数据
RD1[31:0]	O	输出 1
RD2[31:0]	O	输出 2

表 4 GRF 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有寄存器被设置成 0x00000000
2	读寄存器	根据输入的寄存器地址读出数据
3	写寄存器	根据输入的地址，把值写入寄存器

3. ALU.v

表 5 ALU 端口说明

信号名	方向	描述
A[31:0]	I	32 位输入 1
B[31:0]	I	32 位输入 2
ALUCtr [2:0]	I	控制信号 001: 或运算 010:加法运算 011:减法运算
ALUOut[31:0]	O	计算结果

表 6 ALU 功能定义

序号	功能名称	功能描述
1	或	A|B

2	减	A-B
3	加	A+B

4. EXT.v

表 7 Ext 端口说明

信号名	方向	描述
In[25:0]	I	26 位输入
EXTOp[1:0]	I	控制信号 00:对输入进行高位符号扩展 01:对输入进行高位零扩展 10:在输入的低 16 位补上零 11:pc31..28||In[25:0] ||0^2
PC8[31:0]	I	PC+8
Out[31:0]	O	32 位输出

表 8 Ext 功能定义

序号	功能名称	功能描述
1	高位符号扩展	高 16 位补 0 或 1，取决于输入的符号位，输入被扩展成 32 位数据
2	高位零扩展	高 16 位补 0，输入被扩展成 32 位数据
3	低位补零	低 16 位补零，输入被扩展成 32 位数据
4	jal 类扩展	pc31..28||In[25:0] ||0^2

5. DM.v

表 9 DM 端口说明

信号名	方向	描述
CLK	I	时钟信号
MemWrite	I	写入数据存储器的控制信号 1:允许写入 0:不允许写入
RESET	I	复位信号 1:复位 0:无效
PC [31:0]	I	写入指令的 PC 地址
A[31:0]	I	读取或写入地址是 A[11:2]
WD[31:0]	I	要写入 RAM 的数据
RD[31:0]	O	读出的数据

表 10 DM 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有数据被设置成 0x00000000
2	读出数据	根据输入的地址读出存储器内的数据
3	写入数据	根据输入的地址，把数据写入到存储器内

6. mux.v
   a. mux2

表 11 mux2 端口说明

信号名	方向	描述
width	parameter	A 和 B 的位宽，默认值为 32 位
A[width-1:0]	I	输入 A
B[width-1:0]	I	输入 B
Select	I	选择信号 0:输出 A 1:输出 B
Out[width-1:0]	O	计算结果

表 12 mux2 功能定义

序号	功能名称	功能描述
1	选择信号	若 Select=0，则 Out=A；若 Select=1，则 Out=B；

   b. mux4

表 13 mux4 端口说明

信号名	方向	描述
width	parameter	A、B、C、D 的位宽，默认值为 32 位
A[width-1:0]	I	输入 A
B[width-1:0]	I	输入 B
C[width-1:0]	I	输入 C
D[width-1:0]	I	输入 D
Select[1:0]	I	选择信号 00:输出 A 01:输出 B 10:输出 C 11:输出 D
Out[width-1:0]	O	计算结果

表 14 mux4 功能定义

序号	功能名称	功能描述
1	选择信号	若 Select=00，则 Out=A；若 Select01，则 Out=B；若 Select=10，则 Out=C；若 Select=11，则 Out=D

7. ProgramCounter.v

表 15 PC 端口说明

信号名	方向	描述
CLK	I	时钟信号
RESET	I	复位信号 1:复位 0:无效
Stall_F	I	阻塞 PC 信号 1:阻塞 PC 0:无效
NPC[31:0]	I	要写入 pc 的地址
PC_F[31:0]	O	取指令的地址

表 16 PC 功能定义

序号	功能名称	功能描述
1	得到指令地址	PC_F 输出下一条指令的地址
2	复位	Reset=1 时，复位至 32’h00003000
3	阻塞	Stall_F 有效时 PC 寄存器复位

8. IM.v

表 17 IM 端口说明

信号名	方向	描述
A[31:0]	I	要读取的下一条指令的地址是 address[11:2]
RD[31:0]	O	取出的指令

表 18 IM 功能定义

序号	功能名称	功能描述
1	取指令	RD 输出要执行的指令

9. FtoDRegister.v

表 19 F/D 级流水寄存器端口说明

信号名	方向	描述
CLK	I	时钟信号
RESET	I	复位信号 1:复位 0:无效
Stall_D	I	阻塞 F/D 级流水线寄存器 1:阻塞 0:无效
IR_F [31:0]	I	F 级指令
PC_F[31:0]	I	F 级 PC
PC4_F [31:0]	I	F 级 PC+4
IR_D [31:0]	O	D 级指令
PC_D [31:0]	O	D 级 PC
PC4_D [31:0]	O	D 级 PC+4

表 20 F/D 级流水寄存器功能定义

序号	功能名称	功能描述
1	阻塞	Stall_D 有效时 F/D 寄存器的值保持不变
2	传递数据至 D 级	把 IR_F,PC_F,PC4_F 从 F 级传递至 D 级
3	复位	RESET 信号有效时所有输出都为 0

10. DtoERegister.v

表 21 D/E 级流水寄存器端口说明

信号名	方向	描述
CLK	I	时钟信号

RESET	I	复位信号 1:复位 0:无效
Flush_E	I	把寄存器清零信号 1:清零 0:无效
IR_D [31:0]	I	D 级指令
RF_RD1 [31:0]	I	GRF 的输出 1
RF_RD2 [31:0]	I	GRF 的输出 2
EXT [31:0]	I	EXT 部件的输出
PC_D[31:0]	I	D 级 PC
PC8_D [31:0]	I	D 级 PC+8
IR_E [31:0]	O	E 级指令
V1_E [31:0]	O	RD1 传递至 E 级
V2_E [31:0]	O	RD2 传递至 E 级
E32_E [31:0]	O	EXT 传递至 E 级
PC_E [31:0]	O	E 级 PC
PC8_E [31:0]	O	E 级 PC+8

表 22 D/E 级流水寄存器功能定义

序号	功能名称	功能描述
1	清零	Flush_E 有效时 D/E 寄存器的值清零
2	传递数据至 E 级	把 IR_F,RF_RD1,RF_RD2,EXT,PC_D,PC8_D 从 D 级传递至 E 级
3	复位	RESET 信号有效时所有输出都为 0

11. EtoMRegister.v

表 23 E/M 级流水寄存器端口说明

信号名	方向	描述
CLK	I	时钟信号
RESET	I	复位信号 1:复位 0:无效
IR_E [31:0]	I	E 级指令
WriteData_E [31:0]	I	E 级写入 DM 的数据
WriteReg_E [4:0]	I	E 级写入 GRF 的地址

ALUOut_E [31:0]	I	E 级 ALUOut
PC_E[31:0]	I	E 级 PC
PC8_E [31:0]	I	E 级 PC+8
IR_M [31:0]	O	E 级指令
ALUOut_M [31:0]	O	M 级 ALUOut
WriteData_M [31:0]	O	M 级写入 DM 的数据
WriteReg_M [4:0]	O	M 级写入 GRF 的地址
PC_M [31:0]	O	M 级 PC
PC8_M [31:0]	O	M 级 PC+8

表 24 E/M 级流水寄存器功能定义

序号	功能名称	功能描述
1	传递数据至 M 级	把 IR_E,PC_E,PC8_E,ALUOut_E,WriteData_E,WriteReg_E 从 E 级传递至 M 级
2	复位	RESET 信号有效时所有输出都为 0

12. MtoWRegister.v

表 25 M/W 级流水寄存器端口说明

信号名	方向	描述
CLK	I	时钟信号
RESET	I	复位信号 1:复位 0:无效
IR_M [31:0]	I	M 级指令
WriteReg_M [4:0]	I	M 级写入 GRF 的地址
DM_ReadData [31:0]	I	M 级 DM 读出的数据
ALUOut_M [31:0]	I	M 级 ALUOut
PC_M[31:0]	I	M 级 PC
PC8_M [31:0]	I	M 级 PC+8
IR_W [31:0]	O	W 级指令
ALUOut_W[31:0]	O	W 级 ALUOut
ReadData_W[31:0]	O	W 级 DM 读出的数据
WriteReg_W [4:0]	O	W 级写入 GRF 的地址
PC_W [31:0]	O	W 级 PC
PC8_W [31:0]	O	W 级 PC+8

表 26 M/W 级流水寄存器功能定义

序号	功能名称	功能描述
1	传递数据至 W 级	把 IR_M,PC_M,PC8_M,ALUOut_M,WriteReg_M,DM_ReadData 从 E 级传递至 M 级
2	复位	RESET 信号有效时所有输出都为 0

三、控制器设计

1. ControlUnit.v

表 27 ControlUnit 端口说明

信号名	方向	描述
Instr [31:0]	I	指令
Equal_D	I	RD1，RD2 是否相等 1:相等 0:不相等
RegWrite	O	控制是否允许写入寄存器堆 1:允许 0:不允许
MemToReg[1:0]	O	控制把什么数据写入寄存器 00:ALUOut_W 01:ReadData_W 10:PC8_W 11:未定
MemWrite[1:0]	O	控制是否允许写入数据存储器及写入类型 00:不允许写入 01:SW 10:SH 11:SB
ALUCtr[2:0]	O	控制 ALU 运算方式 000:A&B 001:A|B 010:A+B 011:A-B 100:Reg[rt]>>Reg[rs] (SRAV) 101~111:未定义
ALUSrc	O	控制选择 ALU 第二个输入 0:WriteData_E 1:E32_E
RegDst[1:0]	O	控制写入寄存器的地址 00:rt_E 01:rd_E 10:5’d31 11:未定
Branch	O	指令是否是跳转指令 1:是 0:不是
PCSrc [2:0]	O	选择 NPC 000:PC4_F 001:PCBranch_D 010:RD1_D 011:EXT32_D 100:RD2_D 101~111:未定义

ExtOp[1:0]	O	控制 EXT 扩展方式 00:对输入进行高位符号扩展 01:对输入进行高位零扩展 10:在输入的低 16 位补上零 11:pc31..28||In[25:0] ||0^2
LdType [2:0]	O	选择 load 类型 000:LW 及其他指令 001:LH 010:LHU 011:LB 100:LBU 101~111:未定义

表 28 ControlUnit 功能定义 a

func	000000	100001	100011	001000	n/a						n/a
op	000000				001101	000100	100011	101011	001111	000011	000010
rt	n/a				n/a	n/a	n/a	n/a	n/a	n/a	n/a
	nop	addu	subu	jr	ori	beq	lw	sw	lui	jal	j
RegWrite	0	1	1	0	1	0	1	0	1	1	0
MemtoReg[1:0]	xx	00	00	xx	00	xx	01	xx	00	10	xx
MemWrite[1:0]	00	00	00	00	00	00	00	01	00	00	00
ALUCtr[2:0]	xxx	010	011	xxx	001	xxx	010	010	001	xxx	xxx
ALUSrc	x	0	0	x	1	x	1	1	1	x	x
RegDst[1:0]	xx	01	01	xx	00	xx	00	xx	00(rt)	10($31)	xx
Branch	0	0	0	0	0	1	0	0	0	0	0
PCSrc[1:0]	000	000	000	010	000	001	000	000	000	011	011
ExtOp[1:0]	xx	xx	xx	xx	01	00	00	00	10	11	11
LdType [2:0]	000	000	000	000	000	000	000	000	000	000	000

表 29 ControlUnit 功能定义 b

func	001001	n/a	000000	n/a	n/a	n/a	n/a	n/a	n/a	000111	n/a
op	000000	000001	111111	101001	101000	100001	100101	100000	100100	000000	000001
rt	n/a	00001	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	10001
	jalr	bgez	bgezalr	sh	sb	lh	lhu	lb	lbu	srav	bgezal
RegWrite	1	0	1	0	0	1	1	1	1	1	1
MemtoReg[1:0]	10	xx	10	xx	xx	01	01	01	01	00	10
MemWrite[1:0]	00	00	00	10	11	00	00	00	00	00	00
ALUCtr[2:0]	xxx	xxx	xxx	010	010	010	010	010	010	100	xxx
ALUSrc	x	x	x	1	1	1	1	1	1	0	x
RegDst[1:0]	01	xx	01	xx	xx	00	00	00	00	01	10
Branch	0	1	0	0	0	0	0	0	0	0	0
PCSrc[1:0]	010	001	000/100	000	000	000	000	000	000	000	000/001
ExtOp[1:0]	xx	00	xx	00	00	00	00	00	00	xx	00
LdType [2:0]	000	000	000	000	000	001	010	011	100	000	000

2. HazardUnit.v

表 30 HazardUnit 端口说明

信号名	方向	描述
IR_D [31:0]	I	D 级指令
IR_E [31:0]	I	E 级指令
IR_M [31:0]	I	M 级指令
IR_W [31:0]	I	W 级指令
Stall_F	O	阻塞 PC 寄存器信号 1:阻塞 0:无效
Stall_D	O	阻塞 F/D 级寄存器信号 1:阻塞 0:无效
Flush_E	O	清零 D/E 级寄存器 1:清零 0:无效

ForwardA_D [1:0]	O	转发至 D 级 RD1 信号 00:RF_RD1_D 01:ALUOut_M 10:PC8_M 11:未定义
ForwardB_D [1:0]	O	转发至 D 级 RD1 信号 00:RF_RD2_D 01:ALUOut_M 10:PC8_M 11:未定义
ForwardA_E [1:0]	O	转发至 D 级 RD1 信号 00:RD1_E 01:ALUOut_M 10:Result_W 11:PC8_M
ForwardB_E [1:0]	O	转发至 D 级 RD1 信号 00:RD2_E 01:ALUOut_M 10:Result_W 11:PC8_M
ForwardB_M	O	转发至 D 级 RD1 信号 0:WriteData_M 1:Result_W

表 31 Tuse 表

IF/ID 当前指令		
指令类型	源寄存器	Tuse
beq	rs/rt	0
car_r	rs/rt	1
car_i	rs	1
load	rs	1
store	rs	1
store	rt	2
J	n/a	n/a
Jal	n/a	n/a
jr	rs	0

表 32 Tnew 表

ID/EX（Tnew）				EX/MEM(Tnew)				MEM/WB(Tnew)			
jal 0/$31	cal_r 1/rd	cal_i 1/rt	load 2/rt	jal 0/$31	cal_r 0/rd	cal_i 0/rt	load 1/rt	jal 0/$31	cal_r 0/rd	cal_i 0/rt	load 0/rt

表 33 暂停矩阵

IF/ID 当前指令			ID/EX（Tnew）			EX/MEM(Tnew)
指令类型	源寄存器	Tuse	cal_r 1/rd	cal_i 1/rt	load 2/rt	load 1/rt
beq	rs/rt	0	暂停	暂停	暂停	暂停
jr	rs	0	暂停	暂停	暂停	暂停
car_r	rs/rt	1			暂停	
car_i	rs	1			暂停	
load	rs	1			暂停	
store	rs	1			暂停	
store	rt	2				

表 34 转发矩阵

						EX/MEM(Tnew)			MEM/WB(Tnew)			
流水级	源寄存器	涉及指令				cal_r 0/rd	cal_i 0/rt	jal 0/$31	cal_r 0/rd	cal_i 0/rt	load 0/rt	jal 0/$31
IF/ID	rs	beq,jr	MFRSD	ForwardA_D	RF.RD1	AO	AO	AO(10)	n/a	n/a	n/a	n/a
	rt	beq	MFRTD	ForwardB_D	RF.RD2	AO	AO	AO(10)	n/a	n/a	n/a	n/a
ID/EX	rs	cal_r,cal_i,ld,st	MFRSE	ForwardA_E	RS_E(V1_E)	AO	AO	AO(11)	M4	M4	M4	M4(10)
	rt	cal_r,st	MFRTE	ForwardB_E	RT_E(V2_E)	AO	AO	AO(11)	M4	M4	M4	M4(10)
EX/MEM	rt	store	MFRTM	ForwardB_M	RT_M(V2_M)	n/a	n/a	n/a	M4	M4	M4	M4(1)
			转发 MUX	控制信号	输入 0							

三、测试程序

- X:产生冲突的前序指令的类型。
- Y:前序指令在哪个阶段与当前指令产生冲突。
- Z:产生冲突的寄存器。

1. 测试 ADDU 指令

用例编号	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
1	R-M-RS	SUBU	MEM	RS	subu $1, $2, $3 addu $4, $1, $2
2	R-M-RT	SUBU	MEM	RT	subu $1, $2, $3 addu $4, $2, $1

3	R-W-RS	SUBU	WRITE	RS	subu $1, $2, $3
					instru 无关
					addu $4, $1, $2

4	R-W-RT	SUBU	WRITE	RT	subu $1, $2, $3
					instru 无关
					addu $4, $2, $1

5	I-M-RS	ORI	MEM	RS	ori $1, $2, 1000 addu $4, $1, $2
6	I-M-RT	ORI	MEM	RT	ori $1, $2, 1000 addu $4, $2, $1
7	I-W-RS	ORI	WRITE	RS	ori $1, $2, 1000 instru 无关 addu $4, $1, $2
8	I-W-RT	ORI	WRITE	RT	ori $1, $2, 1000 instru 无关 addu $4, $2, $1
9	LD-M-RS	LW	MEM	RS	lw $1, 0($2) addu $4, $1, $2
10	LD-M-RT	LW	MEM	RT	lw $1, 0($2) addu $4, $2, $1

11	LD-W-RS	LW	WRITE	RS	lw $1, 0($2)
					instru 无关
					addu $4, $1, $2

12	LD-W-RT	LW	WRITE	RT	lw $1, 0($2)
					instru 无关
					addu $4, $2, $1

13	JAL-M-RS	JAL	MEM	RS	jal next addu $1,$31,$2
14	JAL-M-RT	JAL	MEM	RT	jal next addu $1,$2,$31
15	JAL-W-RS	JAL	WRITE	RS	jal next instru 无关

					addu $1,$31,$2
16	JAL-W-RT	JAL	WRITE	RT	jal next instru 无关 addu $1,$2,$31

2. 测试 ORI 指令

用例编号	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
1	R-M-RS	SUBU	MEM	RS	subu $1, $2, $3 ori $4,$1,100
2	R-W-RS	SUBU	WRITE	RS	subu $1, $2, $3 instru 无关 ori $4,$1,100
3	I-M-RS	LUI	MEM	RS	lui $1,100 ori $3,$1,100
4	I-W-RS	LUI	MEM	RS	lui $1,100 instru 无关 ori $3,$1,100
5	LD-M-RS	LW	MEM	RS	lw $1,0($2) ori $3, $1, 100
6	LD-W-RS	LW	MEM	RS	lw $1,0($2) instru 无关 ori $3, $1, 100
7	JAL-M-RS	JAL	MEM	RS	jal next ori $1,$31,100
8	JAL-W-RS	JAL	MEM	RS	jal next instru 无关 ori $1,$31,100

3. 测试 LW 指令

用例编号	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
1	R-M-RS	SUBU	MEM	RS	subu $1, $2, $3 lw $4,0($1)

2	R-W-RS	SUBU	WRITE	RS	subu $1, $2, $3
					instru 无关
					lw $4,0($1)

3	I-M-RS	LUI	MEM	RS	lui $1,100 lw $2,0($1)
4	I-W-RS	LUI	MEM	RS	lui $1,100 instru 无关 lw $3,0($1)
5	LD-M-RS	LW	MEM	RS	lw $1,0($2) lw $3,0($1)

6	LD-W-RS	LW	MEM	RS	lw $1,0($2)
					instru 无关
					lw $3, 0($1)

7	JAL-M-RS	JAL	MEM	RS	jal next lw $1,0($31)
8	JAL-W-RS	JAL	MEM	RS	jal next instru 无关 lw $1,0($31)

4. 测试 SW 指令

用例编号	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
1	R-M-RS	SUBU	MEM	RS	subu $1, $2, $3 sw $4,0($1)

2	R-W-RS	SUBU	WRITE	RS	subu $1, $2, $3
					instru 无关
					sw $4,0($1)

3	R-W-RT	SUBU	WRITE	RT	subu $1, $2, $3 sw $1,0($4)

4	R-W-RT	SUBU	WRITE	RT	subu $1,$2,$3
					instru 无关
					sw $1,0($4)

5	I-M-RS	LUI	MEM	RS	lui $1,100 sw $2,0($1)
6	I-W-RS	LUI	WRITE	RS	lui $1,100 instru 无关 sw $3,0($1)
7	I-W-RT	LUI	WRITE	RT	lui $1,100 sw $1,0($2)
8	I-W-RT	LUI	WRITE	RT	lui $1,100 instru 无关 sw $1,0($3)
9	LD-M-RS	LW	MEM	RS	lw $1,0($2) sw $3,0($1)

10	LD-W-RS	LW	WRITE	RS	lw $1,0($2)
					instru 无关
					sw $3, 0($1)

11	LD-W-RT	LW	WRITE	RT	lw $1,0($2) sw $1,0($3)

12	LD-W-RT	LW	WRITE	RT	lw $1,0($2)
					instru 无关
					sw $1,0($3)

13	JAL-M-RS	JAL	MEM	RS	jal next sw $1,0($31)
14	JAL-W-RS	JAL	WRITE	RS	jal next instru 无关 sw $1,0($31)
15	JAL-W-RT	JAL	WRITE	RT	jal next sw $31,0($1)
16	JAL-W-RT	JAL	WRITE	RS	jal next instru 无关 sw $31,0($1)

5. 测试 BEQ 指令

用例编号	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
1	R-E-RS	SUBU	EXECUTE	RS	subu $1, $2, $3 beq $1,$2,next
2	R-M-RS	SUBU	MEM	RS	subu $1, $2, $3 instru 无关 beq $1,$2,next
3	R-W-RS	SUBU	WRITE	RS	subu $1, $2, $3 instru 无关 instru 无关 beq $1,$2,next
4	R-E-RT	SUBU	EXECUTE	RT	subu $1, $2, $3 beq $4,$1,next
5	R-M-RT	SUBU	MEM	RT	subu $1, $2, $3 instru 无关 beq $4,$1,next
6	R-W-RT	SUBU	WRITE	RT	subu $1, $2, $3 instru 无关 instru 无关 beq $4,$1,next
7	I-E-RS	LUI	EXECUTE	RS	lui $1,100 beq $1,$2,next
8	I-M-RS	LUI	MEM	RS	lui $1,100 instru 无关 beq $1,$2,next
9	I-W-RS	LUI	WRITE	RS	lui $1,100 instru 无关 instru 无关 beq $1,$2,next
10	I-E-RT	LUI	EXECUTE	RT	lui $1,100 beq $2,$1,next
11	I-M-RT	LUI	MEM	RT	lui $1,100 instru 无关 beq $2,$1,next
12	I-W-RT	LUI	WRITE	RT	lui $1,100 instru 无关 instru 无关 beq $2,$1,next
13	LD-E-RS	LW	EXECUTE	RS	lw $1,0($2) beq $1,$3,next
14	LD-M-RS	LW	MEM	RS	lw $1,0($2) instru 无关 beq $1,$3,next
15	LD-W-RS	LW	WRITE	RS	lw $1,0($2) instru 无关 instru 无关 beq $1,$3,next
16	LD-E-RT	LW	EXECUTE	RT	lw $1,0($2) beq $3,$1,next

17	LD-M-RT	LW	MEM	RT	lw $1,0($2) instru 无关 beq $3,$1,next
18	LD-W-RT	LW	WRITE	RT	lw $1,0($2) instru 无关 instru 无关 beq $3,$1,next
19//不合法	JAL-E-RS	JAL	EXECUTE	RS	jal next beq $31,$1,next
20	JAL-M-RS	JAL	MEM	RS	jal next instru 无关 beq $31,$1,next
21	JAL-W-RS	JAL	WRITE	RS	jal next instru 无关 instru 无关 beq $31,$1,next
22//不合法	JAL-E-RT	JAL	EXECUTE	RT	jal next beq $1,$31,next
23	JAL-M-RT	JAL	MEM	RT	jal next instru 无关 beq $1,$31,next
24	JAL-W-RT	JAL	WRITE	RT	jal next instru 无关 instru 无关 beq $1,$31,next

6. 测试 JR 指令

用例编号	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
1	R-E-RS	SUBU	EXECUTE	RS	subu $1, $2, $3 jr $1
2	R-M-RS	SUBU	MEM	RS	subu $1, $2, $3 instru 无关 jr $1
3	R-W-RS	SUBU	WRITE	RS	subu $1, $2, $3 instru 无关 instru 无关 jr $1
4	I-E-RS	LUI	EXECUTE	RS	lui $1,100 jr $1
5	I-M-RS	LUI	MEM	RS	lui $1,100 instru 无关 jr $1
6	I-W-RS	LUI	WRITE	RS	lui $1,100 instru 无关 instru 无关 jr $1
7	LD-M-RS	LW	EXECUTE	RS	lw $1,0($2) jr $1

8	LD-M-RS	LW	MEM	RS	lw $1,0($2) instru 无关 jr $1
9	LD-W-RS	LW	WRITE	RS	lw $1,0($2) instru 无关 instru 无关 jr $1
10//未定义行为	JAL-E-RS	JAL	EXECUTE	RS	jal next jr $31
11	JAL-M-RS	JAL	MEM	RS	jal next instru 无关 jr $31
12	JAL-W-RS	JAL	WRITE	RS	jal next instru 无关 instru 无关 jr $31

1. 测试 ADDU 指令

```
.text
main:ori  $t2,100
     ori  $t3,666
     subu $t1,$t2,$t3
     addu $t4,$t1,$t2

     subu $t1,$t2,$t3
     addu $t4,$t2,$t1

     subu $t1,$t2,$t3
     nop
     addu $t4,$t1,$t2

     subu $t1,$t2,$t3
     nop
     addu $t4,$t2,$t1

     ori  $t1,$t2,1000
     addu $4,$1,$2

     ori  $t1,$t2,5678
     addu $t4,$t2,$t1

     ori  $t1,$t2,1000
     nop
     addu $t4,$t1,$t2

     ori  $t1,$t2,1111
     nop
```

```
      addu $t4,$t2,$t1

      sw   $t1,4($0)
      lw   $t1,4($t2)
      addu $t4,$t1,$t2

      lw   $t1,4($t2)
      addu $t4,$t2,$t1

      lw   $t1,4($t2)
      nop
      addu $t4,$t1,$t2

      lw   $t1,4($t2)
      nop
      addu $t4,$t2,$t1

      jal  next
      addu $t1,$31,$2

      jal  next
      addu $t1,$t2,$31

      jal  next
      nop
      addu $t1,$31,$2

      jal  next
      nop
      addu $t1,$t2,$31

      j    end
      nop

next:jr   $31
      nop

end: nop
      nop
```

期望测试结果：

40@00003000: $10 <= 00000064
50@00003004: $11 <= 0000029a
60@00003008: $ 9 <= fffffdca
70@0000300c: $12 <= fffffe2e

80@00003010: $ 9 <= fffffdca
90@00003014: $12 <= fffffe2e
100@00003018: $ 9 <= fffffdca
120@00003020: $12 <= fffffe2e
130@00003024: $ 9 <= fffffdca
150@0000302c: $12 <= fffffe2e
160@00003030: $ 9 <= 000003ec
170@00003034: $ 4 <= 00000000
180@00003038: $ 9 <= 0000166e
190@0000303c: $12 <= 000016d2
200@00003040: $ 9 <= 000003ec
220@00003048: $12 <= 00000450
230@0000304c: $ 9 <= 00000477
250@00003054: $12 <= 000004db
255@00003058: *00000004 <= 00000477
270@0000305c: $ 9 <= 00000000
290@00003060: $12 <= 00000064
300@00003064: $ 9 <= 00000000
320@00003068: $12 <= 00000064
330@0000306c: $ 9 <= 00000000
350@00003074: $12 <= 00000064
360@00003078: $ 9 <= 00000000
380@00003080: $12 <= 00000064
390@00003084: $31 <= 0000308c
400@00003088: $ 9 <= 0000308c
430@0000308c: $31 <= 00003094
440@00003090: $ 9 <= 000030f8
470@00003094: $31 <= 0000309c
510@0000309c: $ 9 <= 0000309c
520@000030a0: $31 <= 000030a8
560@000030a8: $ 9 <= 0000310c

2. 测试 ORI 指令

```
.text

main: ori   $t2,100

      lui   $t3,666

      subu  $t1,$t2,$t3

      ori   $t4,$t1,100


      subu  $t1,$t2,$t3

      nop
```

```
      ori   $t4,$t1,100

      lui   $t1,100
      ori   $t3,$t1,100

      lui   $t1,100
      nop
      ori   $t3,$t1,100

      sw    $t1,4($t0)
      lw    $t1,4($t0)
      ori   $t3,$t1,100

      lw    $t1,4($t0)
      nop
      ori   $t3,$t1,100

      jal   next
      ori   $t1,$31,100

      jal   next
      nop

      j     end

next: ori   $t1,$31,100
      jr    $31
      nop

end:  nop
```

```
      nop
```

期望测试结果：

40@00003000: $10 <= 00000064
50@00003004: $11 <= 029a0000
60@00003008: $ 9 <= fd660064
70@0000300c: $12 <= fd660064
80@00003010: $ 9 <= fd660064
100@00003018: $12 <= fd660064
110@0000301c: $ 9 <= 00640000
120@00003020: $11 <= 00640064
130@00003024: $ 9 <= 00640000
150@0000302c: $11 <= 00640064
155@00003030: *00000004 <= 00640000
170@00003034: $ 9 <= 00640000
190@00003038: $11 <= 00640064
200@0000303c: $ 9 <= 00640000
220@00003044: $11 <= 00640064
230@00003048: $31 <= 00003050
240@0000304c: $ 9 <= 00003074
250@0000305c: $ 9 <= 00003074
280@00003050: $31 <= 00003058
300@0000305c: $ 9 <= 0000307c
340@0000305c: $ 9 <= 0000307c

3. 测试 LW 指令

```
.text
main: ori   $t2,4
      ori   $t3,2
      subu  $t1,$t2,$t3
      lw    $t4,2($t1)

      subu  $t1,$t2,$t3
      nop
      lw    $t4,2($t1)
```

```
      ori   $t1,$t1,4
      lw    $t2,2($t1)

      ori   $t1,$t1,4
      nop
      lw    $t2,2($t1)

      lw    $t1,0($t2)
      lw    $t3,0($t1)

      lw    $t1,0($t2)
      nop
      lw    $t3,0($t1)

      jal   next
      lw    $t1,-0x304c($31)

      jal   next
      nop

      j     end
      nop

next: lw    $t1,-0x304c($31)
      jr    $31
      nop
end:  nop
      nop
```

期望测试结果：

40@00003000: $10 <= 00000004
50@00003004: $11 <= 00000002
60@00003008: $ 9 <= 00000002
70@0000300c: $12 <= 00000000
80@00003010: $ 9 <= 00000002
100@00003018: $12 <= 00000000
110@0000301c: $ 9 <= 00000006
120@00003020: $10 <= 00000000
130@00003024: $ 9 <= 00000006
150@0000302c: $10 <= 00000000
170@00003030: $ 9 <= 00000000
190@00003034: $11 <= 00000000
200@00003038: $ 9 <= 00000000
220@00003040: $11 <= 00000000
230@00003044: $31 <= 0000304c
240@00003048: $ 9 <= 00000000
250@0000305c: $ 9 <= 00000000
280@0000304c: $31 <= 00003054
300@0000305c: $ 9 <= 00000000

4. 测试 SW 指令

```
.text

main: ori   $t2,2

      ori   $t3,2

      ori   $t4,2


      subu  $t1,$t2,$t3

      sw    $t4,4($t1)


      subu  $t1,$t2,$t3

      nop

      sw    $t4,4($t1)


      subu  $t1,$t2,$t3

      sw    $t1,2($t4)
```

```
subu  $t1,$t2,$t3
nop
sw    $t1,2($t4)

lui   $t1,1
sw    $t2,-0x00010000($t1)

lui   $t1,1
nop
sw    $t3,-0x0000FFF8($t1)

lui   $t1,100
sw    $t1,2($t2)

lui   $t1,100
nop
sw    $t1,2($t2)

lw    $t1,2($t2)
sw    $t3,-0x640000($t1)

lw    $t1,2($t2)
nop
sw    $t3,-0x640000($t1)

lw    $t1,2($t2)
sw    $t1,2($t3)

lw    $t1,2($t2)
nop
```

```
      sw    $t1,2($t3)

      jal   next1
      sw    $t1,-0x30a0($31)

      jal   next1
      nop

      jal   next2
      sw    $31,-0x640000($t1)

      jal   next2
      nop

      j     end
      nop
next1:sw    $t1,-0x30a0($31)
next2:sw    $31,-0x640000($t1)
end:  nop
      nop
```

期望测试结果：

40@00003000: $10 <= 00000002
50@00003004: $11 <= 00000002
60@00003008: $12 <= 00000002
70@0000300c: $ 9 <= 00000000
75@00003010: *00000004 <= 00000002
90@00003014: $ 9 <= 00000000
105@0000301c: *00000004 <= 00000002
120@00003020: $ 9 <= 00000000
125@00003024: *00000004 <= 00000000
140@00003028: $ 9 <= 00000000

155@00003030: *00000004 <= 00000000
170@00003034: $ 9 <= 00010000
180@00003038: $ 1 <= ffff0000
190@0000303c: $ 1 <= 00000000
195@00003040: *00000000 <= 00000002
210@00003044: $ 9 <= 00010000
230@0000304c: $ 1 <= ffff0000
240@00003050: $ 1 <= 00000000
245@00003054: *00000008 <= 00000002
260@00003058: $ 9 <= 00640000
265@0000305c: *00000004 <= 00640000
280@00003060: $ 9 <= 00640000
295@00003068: *00000004 <= 00640000
310@0000306c: $ 9 <= 00640000
320@00003070: $ 1 <= ff9c0000
330@00003074: $ 1 <= 00000000
335@00003078: *00000000 <= 00000002
350@0000307c: $ 9 <= 00640000
370@00003084: $ 1 <= ff9c0000
380@00003088: $ 1 <= 00000000
385@0000308c: *00000000 <= 00000002
400@00003090: $ 9 <= 00640000
405@00003094: *00000004 <= 00640000
420@00003098: $ 9 <= 00640000
435@000030a0: *00000004 <= 00640000
450@000030a4: $31 <= 000030ac
455@000030a8: *0000000c <= 00640000
465@000030d4: *0000000c <= 00640000
480@000030d8: $ 1 <= ff9c0000
490@000030dc: $ 1 <= 00000000
495@000030e0: *00000000 <= 000030ac

5. 测试 BEQ 指令

```
.text

main: ori   $t2,4000

      ori   $t3,2000

      subu  $t1,$t2,$t3

      beq   $t1,$t3,next1

      nop


next1:      subu  $t1,$t2,$t3
```

```
      nop
      beq   $t1,$t3,next2
      nop

next2:     subu  $t1,$t2,$t3
      nop
      nop
      beq   $t1,$t3,next3
      nop
next3:     ori   $t4,2000
      subu  $t1,$t2,$t3
      beq   $t4,$t1,next4
      nop
next4:     subu  $t1,$t2,$t3
      nop
      beq   $t4,$t1,next5
      nop
next5:     subu  $t1,$t2,$t3
      nop
      nop
      beq   $t4,$t1,next6
      nop
next6:     lui   $t2,100
      lui   $t1,100
      beq   $t1,$t2,next7
      nop
next7:     lui   $t1,200
      nop
      beq   $t1,$t2,next8
      nop
```

```
      lui   $t1,100
      nop
      nop
      beq   $t1,$t2,next8
      nop
next8:     lui   $t2,100
      beq   $t2,$t1,next9
      nop
next9:     lui   $t2,200
      nop
      beq   $t2,$t1,next10
      nop

      lui   $t2,200
      nop
      nop
      beq   $t2,$t1,next10
      nop
next10:    sw    $t3,96($0)
      sw    $t2,100($0)
      lw    $t2,100($0)
      beq   $t1,$t3,next11
      nop

      lw    $t1,96($0)
      nop
      beq   $t1,$t3,next11
      nop
next11:    lw    $t1,100($0)
```

```
      nop
      nop
      beq   $t1,$t3,next12
      nop

      lw    $t1,100($0)
      beq   $t3,$t1,next12
      nop

      lw    $t1,100($0)
      nop
      beq   $t3,$t1,next12
      nop

      lw    $t1,96($0)
      nop
      nop
      beq   $t3,$t1,next12
      nop
next12:   jal   next13
      nop

      jal   next14
      nop

      jal   next15
      nop

      jal   next16
      nop
```

```
next13:    beq   $31,$t1,next14
      nop
      jr    $31
      nop

next14:    nop
      beq   $31,$t1,next15
      nop
      jr    $31
      nop

next15:    beq   $1,$31,next16
      nop
      jr    $31
      nop

next16:    nop
      beq   $1,$31,end
      nop
end:  nop
      nop
```

期望测试结果：

40@00003000: $10 <= 00000fa0
50@00003004: $11 <= 000007d0
60@00003008: $ 9 <= 000007d0
100@00003014: $ 9 <= 000007d0
140@00003024: $ 9 <= 000007d0
190@00003038: $12 <= 000007d0
200@0000303c: $ 9 <= 000007d0
240@00003048: $ 9 <= 000007d0
280@00003058: $ 9 <= 000007d0
330@0000306c: $10 <= 00640000
340@00003070: $ 9 <= 00640000
380@0000307c: $ 9 <= 00c80000

420@0000308c: $ 9 <= 00640000
470@000030a0: $10 <= 00640000
510@000030ac: $10 <= 00c80000
550@000030bc: $10 <= 00c80000
595@000030d0: *00000060 <= 000007d0
605@000030d4: *00000064 <= 00c80000
620@000030d8: $10 <= 00c80000
650@000030e4: $ 9 <= 000007d0
700@000030f4: $ 9 <= 00c80000
750@00003108: $ 9 <= 00c80000
800@00003114: $ 9 <= 00c80000
850@00003124: $ 9 <= 000007d0
900@00003138: $31 <= 00003140
960@00003140: $31 <= 00003148
1030@00003148: $31 <= 00003150
1090@00003150: $31 <= 00003158

6. 测试 JR 指令

```
.text
main:     ori  $t2,1010
     lui  $t3,678
     jal  next1
     nop
     jal  next2
     nop
     j    end
next1:    jr   $31
     nop
next2:    nop
     jr   $31
     subu $1,$2,$3
end: nop
     nop
```

期望测试结果：

40@00003000: $10 <= 000003f2
50@00003004: $11 <= 02a60000
60@00003008: $31 <= 00003010
100@00003010: $31 <= 00003018
140@0000302c: $ 1 <= 00000000

思考题

1．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？

如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。

此思考题请同学们结合自己测试 CPU 使用的具体手段，按照自己的实际情况进行回答

在实验中，我遇到许多指令产生冲突，解决方案是转发和暂停。我根据教程建 Tuse，Tnew 的表格，利用 Tuse，Tnew 来判断哪些指令应该转发，哪些应该暂停。我的测试指令的构造策略是按照工程化的方法，把冲突表格建好，然后依次测试所有冲突，这样就能测试到所有的冲突情况。