CPU 设计文档

目录：

一、工程化方法与流水线设计

1.指导性原则

（1）规范化命名：对于数据信号，明确指出信号名、阶段名、方向。

（2）模块化结构：把每一个流水级作为一个独立模块，每一级流水线寄存器也作为一个独立模块，数据只能在相邻模块之间传递。

（3）分布式译码：只把指令在流水级之间传递，然后在不同流水级分别实例化一个同样的控制模块进行译码。

2.设计说明

在上次实验的基础上，使用 Verilog 语言设计一个流水线处理器。

（1）支持 MIPS-lite2 指令集：{addu, subu, ori, lw, sw, beq, lui, j, jal, jr, nop}。

（2）addu,subu 不支持溢出。

（3）处理器为流水线设计。

（4）考虑延迟槽。

3.设计要求

（1）顶层视图中将控制器与冲突单元分离，定义独立 controller 模块和 hazard 模块。控制器的设计与单周期没有差别。

（2）流水线的设计以追求性能为第一目标，尽最大可能支持转发以解决数据冒险。

（3）对于 b 类和 j 类指令，流水线设计必须支持延迟槽，因此设计需要注意使用 PC+8。

（4）为了解决数据冒险而设计的转发数据来源必须是某级流水线寄存器，不允许对功能部件的输出直接进行转发。

二、模块设计（沿用单周期模块）

1. IFU（取指令单元）

（1）PC（程序计数器）

PC 用[31:0] reg 实现，具有复位功能。起始地址：0x00003000。

（2）IM（指令存储器）

IM 用[31:0] reg [1023:0]实现，容量为 32bit×1024。以读取文件的方式将 code.txt 中指令加载至 IM 中。

由于 IM 地址宽度为 10 位，PC 为 32 位且起始地址为 0x00003000，故通过 PC 取 IM 中指令时需要选择 PC 的 2-11 位。

（3）端口定义与功能描述

表 2.1.1 IFU 模块端口定义

信号名	方向	描述
NPC[31:0]	I	由 NPC 计算出的下一个 PC 的值
clk	I	时钟信号
reset	I	同步复位信号
branch	I	判断当前指令是需要跳转
Br[1:0]	I	判断分支/跳转指令的 PC 计算方式
PC[31:0]	O	当前 PC 的值
PC4[31:0]	O	当前 PC+4 的值
PC8[31:0]	O	当前 PC+8 的值
Instr[31:0]	O	输出当前 PC 所指的指令

表 2.1.2 IFU 模块功能描述

序号	功能描述	描述
1	复位	时钟上升沿时，若 reset 信号有效，PC 指向 0x0000_3000
2	取指令	根据 PC 取出 IM 中相应指令并输出
3	计算 PC4	计算 PC+4 的值并输出
4	计算 PC8	计算 PC+8 的值并输出
5	更新 PC	通过 branch 确定下一条指令的 PC

2.NPC（计算下一 PC）

表 2.2.1 NPC 模块端口定义

信号名	方向	描述
Instr[31:0]	I	当前 PC 所指的指令
rs[31:0]	I	由 GRF 输出用于 jr 指令的 32 位数值
Br[1:0]	I	控制信号，确定下一指令的计算方法
PC[31:0]	I	当前执行指令地址
NPC[31:0]	O	下一条应执行的指令地址

表 2.2.2 NPC 模块功能描述

序号	功能描述	描述
1	分支指令 beq	跳转，NPC = PC + 4 + {14{ Instr [15]}，Instr [15:0]，0，0}
2	跳转指令 jal	跳转，NPC = {PC[31:28] ，Instr [25:0]，0，0}
3	跳转指令 jr	跳转，NPC = rs

3.CMP（beq 比较前移）

比较两输入，相等时输出，不相等时输出 0。

表 2.3.1 CMP 模块端口定义

信号名	方向	描述
A[31:0]	I	第一个待比较值
B[31:0]	I	第二个待比较值
C	O	比较结果

4. GRF（寄存器堆）

用具有写使能的寄存器实现，寄存器总数为 32 个，0 号寄存器的值始终保持为 0，其他寄存器初始值均为 0。

表 2.4.1 GRF 模块端口定义

信号名	方向	描述
clk	I	时钟信号
reset	I	同步复位信号，将 32 个寄存器中的值全部清零
WE	I	写使能信号。
A1[4:0]	I	地址输入信号，指定 32 个寄存器中的一个，将其中存储

		的数据读出到 RD1
A2[4:0]	I	地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
A3[4:0]	I	地址输入信号，指定 32 个寄存器中的一个，作为写入的目标寄存器
WD[31:0]	I	32 位数据输入信号
RD1[31:0]	O	输出 A1 指定的寄存器中的 32 位数据
RD2[31:0]	O	输出 A2 指定的寄存器中的 32 位数据

表 2.4.2 GRF 模块功能描述

序号	功能描述	描述
1	复位	时钟上升沿时，若 reset 信号有效，所有寄存器数值清零
2	读数据	读出 A1，A2 地址对应寄存器中所存储的数据到 RD1，RD2
3	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入 A3 所对应的寄存器中

5. ALU（算术逻辑单元）

提供 32 位加、减、或运算及大小比较功能，其中加减运算不检测溢出。

表 2.5.1 ALU 模块端口定义

信号名	方向	描述
A[31:0]	I	ALU 的第一个操作数
B[31:0]	I	ALU 的第二个操作数
ALUOp[1:0]	I	控制 ALU 的计算种类
C[31:0]	O	ALU 的运算结果

表 2.5.2 ALU 模块功能描述

序号	功能描述	描述
1	加运算	C = A + B
2	减运算	C = A - B
3	或运算	C = A | B

6. DM（数据存储器）

使用[31:0] reg [1023:0]实现，容量为 32bit×1024。起始地址：0x00000000。

表 2.6.1 DM 模块端口定义

信号名	方向	描述
A[9:0]	I	指向 DM 的地址
Input[31:0]	I	将要写入 DM 的数据
clk	I	时钟信号
reset	I	异步复位信号
MemWr	I	DM 写使能信号
D[31:0]	O	DM 输出的数据

表 2.6.2 DM 模块功能描述

序号	功能描述	描述
1	复位	时钟上升沿时，若 reset 信号有效，所有寄存器数值清零
2	读数据	读出 A 地址中所存储的数据到 D
3	写数据	当 MemWr 信号有效且时钟上升沿来临时，将 Input 输入的数据写入 A 所对应的地址中

7. EXT（位扩展器）

表 2.7.1 EXT 模块端口定义

信号名	方向	描述
A[15:0]	I	待扩展数
EXTOp[1:0]	I	控制扩展类型
B[31:0]	O	输出扩展结果

表 2.7.2 EXT 模块功能描述

序号	功能描述	描述
1	高位 0 扩展	B = {16{0}，A}
2	低位 0 扩展	B = {A，16{0}}
3	符号扩展	B = {16{A[15]}，A}

三、数据通路设计

1.基础数据通路设计

（1）构造表头：表头的内容，主要包括各个阶段包含的部件、各个流水级寄存器以及它们输入的数据，依据它们来构造数据通路。在表头中，我们写出每一个功能模块以及这些功能模块的每个数据输入。对于 RF 我们把读功能和写功能分别写在 D 级和 W 级。

（2）分析指令，并填写表格：基础数据通路中暂且忽略流水线带来的各种冒险，只考虑数据从哪个部件流动到哪个部件，最后结果去向何处，最终写在表格中。由于数据不能跨越流水级寄存器进行传递，所以某个端口如果在这条指令的执行过程中不需要使用则留空。

（3）构造多选器：有些端口有着多个可能的数据来源，我们需要对这些输入端口前加入一个多选器，来控制数据来源的选择。

2.暂停机制控制

当判定需要暂停的时候，我们需要执行三个操作：

（1）冻结 PC，不让 PC 的值改变——IFU 增加 PC 写使能信号(!stall)。

（2）让 ID/EX 流水级寄存器清零——E 级流水寄存器增加 clr 信号(stall)。

（3）冻结 IF/ID 流水级寄存器——D 级寄存器增加写使能信号(!stall)。

3.转发机制控制

分析转发情况，构造转发多选器：每一个数据需求，它的正确数据，可能是之前从寄存器堆中读取来的值，也有可能是其他流水级尚未来得及写入寄存器堆的结果。

因此构造转发表格，分析冲突情况，构造转发多选器，把转发多选器加入数据通路中，控制信号由控制器输入。

表 3.3.1 转发多选器

MUX	控制信号	输入来源				
MF_RD1_D	RD1_D	RF.RD1	PC8@M	AO@M	PC8@E	
MF_RD2_D	RD2_D	RF.RD2	PC8@M	AO@M	PC8@E	
MF_RD1_E	RD1_E	RD1@E	WD		PC8@M	AO@M
MF_RD2_E	RD2_E	RD2@E	WD		PC8@M	AO@M
MF_RD2_M	RD2_M	RD2@M	WD			

当 W 级需要向 D 级转发时，由于 GRF 同时具有读写功能，可以在 GRF 模块内判断是否需要转发并实现转发，不通过 datapath 中的转发多选器实现。

4.设计数据通路

表 3.4.1 datapath 模块端口定义

信号名	方向	描述
reset	I	同步复位信号
clk	I	时钟信号
branch	I	是否为分支/跳转指令的判断信号
Br[1:0]	I	分支/跳转地址计算方法的判断信号
RegIn,	I	GRF 输入地址来源控制信号
RegWr,	I	GRF 写使能信号
EXTOp[1:0]	I	EXT 扩展方式控制信号
ALUsrc,	I	ALU 第二个运算数来源控制信号
ALUOp[1:0]	I	ALU 计算方式控制信号
MemWr	I	DM 写使能信号
MemtoReg	I	GRF 输入来源控制信号
RES[2:0]	I	AT 编码器计算出的 Tnew 信号
stall	I	冲突检测器传入暂停信号
MRD1_D[2:0]	I	转发控制器传入 MF_RD1_D 的选择信号
MRD2_D[2:0]	I	转发控制器传入 MF_RD2_D 的选择信号
MRD1_E[2:0]	I	转发控制器传入 MF_RD1_E 的选择信号
MRD2_E[2:0]	I	转发控制器传入 MF_RD2_E 的选择信号
MRD2_M[2:0]	I	转发控制器传入 MF_RD2_M 的选择信号
IR_D[31:0]	O	D 级指令
IR_E[31:0]	O	E 级指令
IR_M[31:0]	O	M 级指令
IR_W[31:0]	O	W 级指令
RES_E[2:0]	O	E 级 RES
RES_M2:0]	O	M 级 RES

RES_W[2:0]	O	W 级 RES
A1_E[4:0]	O	E 级 A1
A2_E[4:0]	O	E 级 A2
A2_M[4:0]	O	M 级 A2
A3_E[4:0]	O	E 级 A3
A3_M[4:0]	O	M 级 A3
A3_W[4:0]	O	W 级 A3

部件	输入	输入来源			MUX	控制	addu	subu	ori	lw	sw	beq	lui	j	jal	jr	nop
IFU	PC	ADD4	NPC		M_PC	branch	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4|NPC	ADD4	NPC	NPC	NPC	
	ADD4	PC					PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	
	ADD8	PC					PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	
	IM	PC					PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	
D级	PC@D	PC					PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	
	IR@D	IM					IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	
	PC8@D	ADD8													ADD8		
RF	A1	IR@D[rs]					IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]			IR@D[rs]	
	A2	IR@D[rt]					IR@D[rt]	IR@D[rt]				IR@D[rt]					
EXT	EXT	IR@D[i16]							IR@D[i16]	IR@D[i16]	IR@D[i16]		IR@D[i16]				
CMP	D1	MF_RD1_D										RF.RD1					
	D2	MF_RD2_D										RF.RD2					
NPC	PC	PC@D										PC@D		PC@D	PC@D		
	I26	IR@D[i16]										IR@D[i16]		IR@D[i26]	IR@D[i26]		
	JR	MF_RD1_D														RF.RD1	
E级	IR@E	IR@D					IR@D	IR@D	IR@D	IR@D	IR@D		IR@D		IR@D		
	PC8@E	PC8@D													PC8@D		
	A1@E	IR@D[rs]					IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]		IR@D[rs]				
	A2@E	IR@D[rt]					IR@D[rt]	IR@D[rt]									
	A3@E	IR@D[rd]	IR@D[rt]	31	M_RF_A3	RegIn	IR@D[rd]	IR@D[rd]	IR@D[rt]	IR@D[rt]			IR@D[rt]		31		
	RD1@E	MF_RD1_D					RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1		RF.RD1				
	RD2@E	MF_RD2_D					RF.RD2	RF.RD2									
	EXT@E	EXT							EXT	EXT	EXT		EXT				
ALU	A	MF_RD1_E					RD1@E	RD1@E	RD1@E	RD1@E	RD1@E		RD1@E				
	B	MF_RD2_E	EXT@E		M_ALU_B	ALUsrc	RD2@E	RD2@E	EXT@E	EXT@E	EXT@E		EXT@E				
M级	IR@M	IR@E					IR@E	IR@E	IR@E	IR@E	IR@E		IR@E		IR@E		
	PC8@M	PC8@E													PC8@E		
	A2@M	A2@E					A2@E	A2@E									
	A3@M	A3@E					A3@E	A3@E	A3@E	A3@E	A3@E		A3@E				
	RD2@M	RD2@E									RD2@E						
	AO@M	ALU					ALU	ALU	ALU	ALU	ALU		ALU				
DM	A	AO@M								AO@M	AO@M						
	WD	MF_RD2_M									RD2@M						
W级	IR@W	IR@M					IR@M	IR@M	IR@M	IR@M	IR@M		IR@M				
	PC8@W	PC8@M													PC8@M		
	A3@W	A3@M					A3@M	A3@M	A3@M	A3@M			A3@M		A3@M		
	AO@W	AO@M					AO@M	AO@M	AO@M				AO@M				
	DR@W	DM								DM							
RF	A3	A3@W					A3@W	A3@W	A3@W	A3@W			A3@W		A3@W		
	WD	AO@W	DR@W	PC8@W	M_WD_src	ToReg	AO@W	AO@W	AO@W	DR@W			AO@W		PC8@W		

图 3.1 数据通路设计表格

四、控制器设计

1.主控制器

主控制器由指令生成相应控制信号，为精简流水寄存器，在每一级流水处设置一个主控制器，由本级指令产生本级所需控制信号。

表 4.1.1 非转发暂停控制信号

控制信号	功能名称
branch	branch = 0 时，PC 端输入为 IFU.PC4； branch = 1 时，PC 端输入为 NPC。
Br	Br = 00 时，该指令为分支指令 beq，NPC = PC + 4 + sign_ext(Imm||00)。 Br = 01 时，该指令为跳转指令 jal 或 j，NPC = {PC[31:28] ，Jal，0，0}。 Br = 10 时，该指令为跳转指令 jr，NPC = rs。
RegIn	RegIn = 00 时，数据写入 rt 寄存器； RegIn = 01 时，数据写入 rd 寄存器。 RegIn = 10 时，数据写入 31 号寄存器。
RegWr	RegWr = 0 时，GRF 不能写入数据； RegWr = 1 时，GRF 可以写入数据。
EXTOp	EXTOp = 00 时，进行高位 0 扩展，B = {16{0}，A}； EXTOp = 01 时，进行低位 0 扩展，B = {A，16{0}}； EXTOp = 10 时，进行符号扩展，B = {16{A[15]}，A}；。
ALUSrc	ALUSrc = 0 时，ALU 的第二个运算数来自转发复选器 MF_RD2_E； ALUSrc = 1 时，ALU 的第二个运算数来自位扩展之后的立即数。
ALUOp	ALUOp = 00 时，进行加运算，C = A + B； ALUOp = 01 时，进行减运算，C = A - B； ALUOp = 10 时，进行或运算，C = A | B；
MemWr	MemWr = 0 时，DM 不能写入数据； MemWr = 1 时，DM 可以写入数据。
ToReg	ToReg = 00 时，写入寄存器的数据来自 ALU； ToReg = 01 时，写入寄存器的数据来自 DM。 ToReg = 01 时，写入寄存器的数据来自 NPC。

表 4.1.2 非转发暂停控制信号产生的真值表

指令	opcode	funct	branch	Br	RegIn	RegWr	EXTOp	ALUSrc	ALUOp	MemWr	ToReg
addu	000000	100001	0	X	01	1	X	0	00	0	00
subu	000000	100011	0	X	01	1	X	0	01	0	00
ori	001101		0	X	00	1	00	1	10	0	00
lw	100011		0	X	00	1	10	1	00	0	01
sw	101011		0	X	X	0	10	1	00	1	X
beq	000100		1	00	X	0	X	0	X	0	X
lui	001111		0	X	00	1	01	1	00	0	00
nop	000000		0	X	00	0	00	0	00	0	00
j	000010		1	01	X	0	X	X	X	0	X
jal	000011		1	01	10	1	X	X	X	0	10
jr	000000	001000	1	10	X	0	X	X	X	0	X

附表 支持指令集编码与操作

指令	编码形式	操作
addu	000000 rs rt rd 00000 100001	GPR[rd] = GPR[rs] + GPR[rt]
subu	000000 rs rt rd 00000 100011	GPR[rd] = GPR[rs] − GPR[rt]
ori	100011 base rt offset	GPR[rt] = GPR[rs] OR zero_extend(immediate)
lw	100011 base rt offset	Addr = GPR[base] + sign_ext(offset) GPR[rt] = memory[Addr]
sw	101011 base rt offset	Addr = GPR[base] + sign_ext(offset) memory[Addr] = GPR[rt]

beq	000100 rs rt offset	if (GPR[rs] == GPR[rt])
		PC = PC + 4 + sign_extend(offset‖00)
		else
		PC = PC + 4

lui	001111 00000 rt immediate	GPR[rt] = immediate‖0^16
j	000010 instr_index	PC = PC[31:28] || instr_index || 00
jr	000000 rs 10{0} 5{0} 001000	PC = GPR[rs]

jal	000011 instr_index	PC = PC[31:28] || instr_index || 00 GPR[31] = PC + 4
nop	0x00000000	

2.AT 编码器

根据指令变量与，产生 Tuse 变量和 Res 编码值（Res 是 Tnew 的变形，用于指定产生新值的功能部件）。

此处将指令分为 Cal_r 类(addu, subu)、Cal_i 类(ori)、Beq 类(beq)、Load 类(lw)、Save 类(sw)分类处理，J 类的三条指令(j, jal, jr)分开处理。

表 4.2.1 指令集的 Tuse

指令	Tuse	
	rs	rt
Cal_r 类	1	1
Cal_i 类	1	
load 类	1	
store 类	1	2
beq	0	0
lui		
nop		
j		
jal		
jr	0	

表 4.2.2 指令集的 Tnew

指令	功能部件	Tnew		
		E	M	W
Cal_r 类	ALU	1	0	0
Cal_i 类	ALU	1	0	0
load 类	DM	2	1	0
store 类				
beq				
lui	ALU	1	0	0
nop				
j				
jal	PC	0	0	0
jr				

3.冲突检测器

根据策略矩阵计算分类指令的暂停条件，根据条件按照逻辑取与和或运算。

表 4.3.1 rs 策略矩阵

Tnew / Tuse	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F

表 4.3.2 rt 策略矩阵

Tnew / Tuse	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F

4.转发控制器

根据转发多选器需求和原则产生转发多选器的控制信号，控制信号的优先级由顺序决定，通过三目运算符的比较赋值顺序实现。

转发条件：转发接受级读寄存器编号与被转发级写寄存器编号相同不为 0，转发级寄存器写使能信号为 1。

转发来源：RES 所标记的被转发级指令对应的新值产生部件储存在流水寄存器中的值。

表 4.4.1 转发控制信号

控制信号	取值	功能说明
MF_RD1_D	MF_PC8_E	向 D 级需要 RD1 值的端口转发 PC8_E
	MF_PC8_M	向 D 级需要 RD1 值的端口转发 PC8_M
	MF_AO_M	向 D 级需要 RD1 值的端口转发 AO_M
	MF_RD	D 级需要 RD1 值的端口无需转发
MF_RD2_D	MF_PC8_E	向 D 级需要 RD2 值的端口转发 PC8_E
	MF_PC8_M	向 D 级需要 RD2 值的端口转发 PC8_M
	MF_AO_M	向 D 级需要 RD2 值的端口转发 AO_M
	MF_RD	D 级需要 RD2 值的端口无需转发
MF_RD1_E	MF_PC8_M	向 E 级需要 RD1 值的端口转发 PC8_M
	MF_AO_M	向 E 级需要 RD1 值的端口转发 AO_M
	MF_WD_W	向 E 级需要 RD1 值的端口转发 W 级的 WD 值
	MF_RD	E 级需要 RD1 值的端口无需转发
MF_RD2_E	MF_PC8_M	向 E 级需要 RD2 值的端口转发 PC8_M
	MF_AO_M	向 E 级需要 RD2 值的端口转发 AO_M
	MF_WD_W	向 E 级需要 RD2 值的端口转发 W 级的 WD 值
	MF_RD	E 级需要 RD2 值的端口无需转发
MF_RD2_M	MF_WD_W	向 M 级需要 RD2 值的端口转发 W 级的 WD 值
	MF_RD	M 级需要 RD2 值的端口无需转发

五、测试程序

1.无分支/跳转测试

表 5.1 CPU 测试程序与期望输出（1）

测试程序	期望输出或现象
1 ori $t0,$t0,0xffff	00003000: $ 8 <= 0000ffff
2 lui $t1,0xffff	00003004: $ 9 <= ffff0000
3 addu $t2,$t0,$t1	00003008: $10 <= ffffffff
4 subu $t3,$t1,$t0	0000300c: $11 <= fffe0001
5 ori $t4,4	00003010: $12 <= 00000004
6 sw $t3,0($t4)	00003014: *00000004 <= fffe0001
7 lw $t5,0($t4)	00003018: $13 <= fffe0001

2. 分支/跳转测试

表 5.2 CPU 测试程序与期望输出（2）

测试程序	期望输出或现象
1 ori $t0,$t0,0xffff	00003000: $ 8 <= 0000ffff
2 beq $zero,$zero,yes	跳转至第 6 行
3 nop	nop
4 lui $t1,0xffff	不执行
5 *yes:*	
6 beq $t0,$zero,noo	不跳转
7 nop	nop
8 ori $t1,$t1,0x1	00003018: $ 9 <= 00000001
9 *noo:*	
10 ori $t2,$t1,0xffff	0000301c: $10 <= 0000ffff

表 5.3 CPU 测试程序与期望输出（3）

测试程序	期望输出或现象
1 ori $t0,$t0,0xffff	00003000: $ 8 <= 0000ffff
2 jal jjal	00003004: $31 <= 0000300c，跳转至 8 行

3 nop	nop
4 lui $t1,0xffff	0000300c: $ 9 <= ffff0000
5 j jjj	跳转至 13 行
6 nop	nop
7 *jjal:*	
8 addu $t2,$t0,$t1	00003018: $10 <= 0000ffff
9 subu $t3,$t1,$t0	0000301c: $11 <= ffff0001
10 jr $ra	跳转至第 4 行
11 nop	nop
12 *jjj:*	
13 ori $t4,4	00003028: $12 <= 00000004

3. 暂停/转发机制测试

表 5.4 CPU 测试程序与期望输出（4）

序号	测试目的	测试程序	期望输出或现象
1	rs0_E1 rt0_E1 MF_RD1_D MF_RD2_D	ori $t0,0xffff beq $t0,$t0,hhh …… *hhh:*	beq 进入 D 后暂停 1 次，跳转 从 M 向 D 转发 AO_M 给 CMP_A 从 M 向 D 转发 AO_M 给 CMP_B,

2	rs0_E2
	rs0_M2
	rt0_E2
	rt0_M2
	MF_RD1_E
	MF_RD2_M
		ori $t0,0xffff
		sw $t0,4($zero)
		lw $t1,4($zero)
		beq $t1,$t1,hhh
		……
		hhh:
			beq 进入 D 后暂停 2 次，不跳转
			从 M 向 E 转发 WD 给 ALU_B
			从 W 向 M 转发 WD 给 DM_A

3	rs1_E2
	rt1_E2
	MF_RD1_E
	MF_RD2_E
		ori $t0,0xffff
		sw $t0,4($zero)
		lw $t1,4($zero)
		addu $t2,$t1,$t1
			addu 进入 D 后暂停 1 次
			从 W 向 E 转发 WD 给 ALU_A
			从 W 向 E 转发 WD 给 ALU_B

4	MF_RD1_D MF_RD2_D	jal hhh nop *hhh:* beq $ra,$ra,emm	从 M 向 D 转发 PC8_M 给 CMP_A 从 M 向 D 转发 PC8_M 给 CMP_B

5	MF_RD1_E
	MF_RD2_E
	GRF 内部转发 W-D
		ori $t0,0xffff
		addu $t1,$t0,$t0
		nop
		addu $t2,$t0,$t0
			从 M 向 E 转发 AO_M 给 ALU_A 和 ALU_B，GRF 内部转发输出$t0

6	MF_RD1_D MF_RD2_D MF_RD1_E MF_RD2_E	jal hhh addu $t0,$ra,$ra …… *hhh:*	从 E 向 D 转发 PC8_E 给 RD1 从 E 向 D 转发 PC8_E 给 RD2 从 M 向 E 转发 PC8_M 给 ALU_A 从 M 向 E 转发 PC8_M 给 ALU_B

六、思考题

Q：在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来。(非常重要)

A：按照 Tuse 和 RES 划分指令，分别有暂停情况 8 种、转发情况 20 种：

```
wire stall_rs0_E1 = ((Tuse_rs == `Tuse_0) && (RES_E == `RES_ALU) && (A1 == A3_E) && (A1 != 0));
wire stall_rs0_E2 = ((Tuse_rs == `Tuse_0) && (RES_E == `RES_DM) && (A1 == A3_E) && (A1 != 0));
wire stall_rs0_M2 = ((Tuse_rs == `Tuse_0) && (RES_M == `RES_DM) && (A1 == A3_M) && (A1 != 0));
wire stall_rs1_E2 = ((Tuse_rs == `Tuse_1) && (RES_E == `RES_DM) && (A1 == A3_E) && (A1 != 0));

wire stall_rt0_E1 = ((Tuse_rt == `Tuse_0) && (RES_E == `RES_ALU) && (A2 == A3_E) && (A2 != 0));
wire stall_rt0_E2 = ((Tuse_rt == `Tuse_0) && (RES_E == `RES_DM) && (A2 == A3_E) && (A2 != 0));
wire stall_rt0_M2 = ((Tuse_rt == `Tuse_0) && (RES_M == `RES_DM) && (A2 == A3_M) && (A2 != 0));
wire stall_rt1_E2 = ((Tuse_rt == `Tuse_1) && (RES_E == `RES_DM) && (A2 == A3_E) && (A2 != 0));
```

图 6.1 暂停情况

```
assign RD1_D = (A1_D == A3_E && RES_E == `RES_PC && A1_D != 0) ? `MF_PC8_E :
               (A1_D == A3_M && RES_M == `RES_PC && A1_D != 0) ? `MF_PC8_M :
               (A1_D == A3_M && RES_M == `RES_ALU && A1_D != 0) ? `MF_AO_M : `MF_RD;
assign RD2_D = (A2_D == A3_E && RES_E == `RES_PC && A2_D != 0) ? `MF_PC8_E :
               (A2_D == A3_M && RES_M == `RES_PC && A2_D != 0) ? `MF_PC8_M :
               (A2_D == A3_M && RES_M == `RES_ALU && A2_D != 0) ? `MF_AO_M : `MF_RD;
assign RD1_E = (A1_E == A3_M && RES_M == `RES_PC && A1_E != 0) ? `MF_PC8_M :
               (A1_E == A3_M && RES_M == `RES_ALU && A1_E != 0) ? `MF_AO_M :
               (A1_E == A3_W && RES_W != `RES_NO && A1_E != 0) ? `MF_WD_W :`MF_RD;
assign RD2_E = (A2_E == A3_M && RES_M == `RES_PC && A2_E != 0) ? `MF_PC8_M :
               (A2_E == A3_M && RES_M == `RES_ALU && A2_E != 0) ? `MF_AO_M :
               (A2_E == A3_W && RES_W != `RES_NO && A2_E != 0) ? `MF_WD_W :`MF_RD;
assign RD2_M = (A2_M == A3_W && RES_W != `RES_NO && A2_M != 0) ? `MF_WD_W :`MF_RD;

assign RD1_GRF = (A1 == A3 && WE && A1 != 0) ? WD : GRF[A1];
assign RD2_GRF = (A2 == A3 && WE && A2 != 0) ? WD : GRF[A2];
```

图 6.2 转发情况

测试样例见第五部分测试程序与期望输出，由于一些冲突会同时发正在一条指令中，因此实际测试样例较为精简。