计算机组成原理 P6 实验报告

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 - CPU，支持的指令集包含{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}。为了实现这些功能，CPU 在顶层模块 mips 下并列包含了 DATAPATH、FORWARD_CONTROL、STOP_CONTROL 三个模块。

DATAPATH 模块下分五个流水级 F、D、E、M、W。F 级包含了 PC、IM 部件和用于 PC 选择的 MUX_PcSel、用于 pc+4 的 add4 两个小部件，D 级包含了 GRF、EXT、NPC、CMP 部件和用于写入寄存器 A3 选择的功能性部件 MUX_RegA3Sel；E 级包含了 ALU 部件、MD 部件（用于计算乘除法）和用于 ALUB 选择的功能性部件 MUX_AluBSel；M 级包含了 DM 部件；W 级连接到 D 级寄存器，包含了用于存入寄存器数据选择的功能性部件 MUX_RegDataSel。相邻两流水级之间还各设置了一个流水线寄存器 regD、regE、regM、regW，用于存储流水的信息。此外，为满足数据冒险的转发需求，还设置了五个转发多路选择器：D 级 MFCMP1D、MFCMP2D 分别用于选择需进入 CMP 部件和 E 级寄存器的两个寄存器值；E 级 MFALUAE、用于选择参与 ALU 运算的第一个数据，MFALUBE 用于选择参与 ALU 运算的第二个来自寄存器的数据和进入 M 级寄存器的数据；MFDM 用于选择写入 DM 的数据。

FORWARD_CONTROL 模块用于生成转发信号控制转发。

STOP_CONTROL 模块用于生成暂停信号控制暂停。

此外，工程文件中包含了名为 define 的.v 文件用于宏定义。本流水线 CPU 主要采用分布式译码，主要将指令与 A3 流水，所有需要指令信息的部件下均包括了译码部件 DECODE，用于产生改位置指令操作所需信息。

（二）关键模块定义

1. DATAPATH

（1） PC

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位至 0x00003000）
en	I	使能信号，为 1 时 PC 正常工作，为 0 时 PC 冻结
pcin[31:0]	I	32 位输入次指令地址
pc[31:0]	O	32 位输出当前指令地址

（2） IM

功能描述	指令存储器，内含 32*1024 字存储器，根据输入的地址输出指令	
信号名	方向	描述
pc[31:0]	I	32 位输入当前指令地址
instr[31:0]	O	32 位输出指令信号

（3） GRF

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
instr[31:0]	I	31 位输入 D 级指令，译码出写使能
instr_W[31:0]	I	31 位输入 W 级指令，译码出写使能
A3_W[4:0]	I	写入寄存器编号
RegData[31:0]	I	回写寄存器的值
pc4[31:0]	I	32 位输入当前 pc+4，用于 display
RD1[31:0]	O	第一个寄存器编号读出的值
RD2[31:0]	O	第二个寄存器编号读出的值

（4） CMP

功能描述	用于前置的 beq 跳转判断	
信号名	方向	描述
D1[31:0]	I	32 位输入用于判断的第一个数据
D2[31:0]	I	32 位输入用于判断的第一个数据
instr[31:0]	O	32 位输入当前指令，用于判断选择比较操作
zero	O	1 位输出判断结果，为 0 表示不跳转，1 表示跳转

（5） EXT

信号名	方向	功能
instr[31:0]	I	32 为输入当前指令，译码出操作选择信号和用于操作的立即数
ext[31:0]	O	32 位输出扩展结果

（6） NPC

信号名	方向	描述
pc4[31:0]	I	32 位输入当前 pc+4
instr[31:0]	I	32 位输入当前指令，译码出操作选择信号
zero	I	辅助选择 NPC（目前仅为 beq 使用）
ra[31:0]	I	32 位输入来自寄存器的指令地址
npc[31:0]	O	32 位输出用于跳转的次指令地址

（7） ALU

信号名	方向	描述
A[31:0]	I	第一个 32 位操作数
B[31:0]	I	第二个 32 位操作数
instr[31:0]	I	32 位输入当前指令，译码出操作选择信号
C[31:0]	O	32 位输出计算结果

（8） MD

信号名	方向	描述
instr[31:0]	I	32 位输入当前指令，译码出操作选择信号
clk	I	时钟信号
reset	I	复位信号
D1[31:0]	I	32 位输入第一个操作数
D2[31:0]	I	32 位输入第二个操作数
HI[31:0]	O	32 位输出 HI 读取结果
LO[31:0]	O	32 位输出 LO 读取结果
start	O	1 位输出表示乘除运算开始
busy	O	1 位输出表示乘除模块工作状态

（9） DM

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
pc4[31:0]	I	32 位输入当前 pc+4
instr[31:0]	I	32 位输入当前指令
Addr[31:0]	I	32 位输入写入 DM 的地址
Data[31:0]	I	32 位输入写入的数据（目前全部来自 rt 寄存器）
DMRD[31:0]	O	32 位输出读出的数据

（10） regD

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
instr[31:0]	I	32 位输入当前指令
pc4[31:0]	I	32 位输入当前 pc+4
en	I	使能信号，1 则寄存器正常工作，0 则冻结寄存器所存内容
instr_D[31:0]	O	32 位输出 D 级指令

pc4_D[31:0]	O	32 位输出 D 级 pc+4

（11） regE

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
clr	I	清空信号，1 则清除寄存器中所有值，0 则寄存器正常工作
instr[31:0]	I	32 位输入当前指令
V1[31:0]	I	32 位输入经转发选择后的 rs 寄存器值
V2[31:0]	I	32 位输入经转发选择后的 rt 寄存器值
ext[31:0]	I	32 位输入 ext 扩展结果
pc4[31:0]	I	32 位输入当前 pc+4
A3[4:0]	I	5 位输入当前指令对应写入寄存器编号
instr_E[31:0]	O	32 位输出 E 级指令
V1_E[31:0]	O	32 位输出 E 级 V1
V2_E[31:0]	O	32 位输出 E 级 V2
ext_E[31:0]	O	32 位输出 E 级 ext
pc4_E[31:0]	O	32 位输出 E 级 pc+4
A3_E[4:0]	O	5 位输出 E 级指令对应的写入寄存器编号

（12） regM

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
instr[31:0]	I	32 位输入当前指令
V2[31:0]	I	32 位输入 V2
ALUC[31:0]	I	32 位输入 ALU 计算结果
pc4[31:0]	I	32 位输入当前 pc+4
HI[31:0]	I	32 位输入 MD 读取 HI 寄存器结果

LO[31:0]	I	32 位输入 MD 读取 LO 寄存器结果
A3[4:0]	I	5 位输入当前指令对应写入寄存器编号
instr_M[31:0]	O	32 位输出 M 级指令
V2_M[31:0]	O	32 位输出 M 级 V2
ALUC_M[31:0]	O	32 位输出 M 级 ALU 计算结果
pc4_M[31:0]	O	32 位输出 M 级 pc+4
HI[31:0]	I	32 位输出 M 级 HI
LO[31:0]	I	32 位输出 M 级 LO
A3_M[4:0]	O	5 位输出 M 级指令对应的写入寄存器编号

（13） regW

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
instr[31:0]	I	32 位输入当前指令
pc4[31:0]	I	32 位输入当前 pc+4
ALUC[31:0]	I	32 位输入 ALU 计算结果
DMRD[31:0]	I	32 位输入 DM 读取结果
HI[31:0]	I	32 位输入 M 级 HI
LO[31:0]	I	32 位输入 M 级 LO
A3[4:0]	I	5 位输入当前指令对应写入寄存器编号
instr_W[31:0]	O	32 位输出 W 级指令
pc4_W[31:0]	O	32 位输出 W 级 pc+4
ALUC_W[31:0]	O	32 位输出 W 级 ALU 计算结果
DMRD_W[31:0]	O	32 位输出 W 级 DM 读取结果
HI[31:0]	I	32 位输出 W 级 HI
LO[31:0]	I	32 位输出 W 级 LO
A3_W[4:0]	O	5 位输出 W 级指令对应的写入寄存器编号

（14） add4

功能描述	用于 pc+4	
信号名	方向	描述
pc[31:0]	I	32 位输入当前指令地址
pc4[31:0]	O	32 位输出 pc+4

（15） MUX_PcSel

功能描述	用于选择使用普通 pc+4 或跳转指令地址	
信号名	方向	描述
pc4[31:0]	I	32 位输入 pc+4
instr[31:0]	I	32 位输入当前指令，译码出选择信号
npc[31:0]	I	32 位输入当前指令，译码用于选择进入 pc 的次指令地址
pcin[31:0]	O	32 位输出进入 pc 的次指令地址

（16） MUX_RegA3Sel

功能描述	用于选择写入寄存器编号	
信号名	方向	描述
instr[31:0]	I	32 位输入当前指令，译码出选择信号
A3[4:0]	O	5 位输出当前指令对应写入寄存器编号

（17） MUX_AluBSel

功能描述	用于选择进入 ALU 计算的 B 数据	
信号名	方向	描述
instr[31:0]	I	32 位输入当前指令，译码出选择信号
V2[31:0]	I	32 位输入经转发选择后的 rt 寄存器数据
ext[31:0]	I	32 位输入 ext 计算结果
ALUB[31:0]	O	32 位输出进入 ALU 的 B 数据

（18） MUX_RegDataSel

功能描述	用于选择存入寄存器的数据	
信号名	方向	描述
instr[31:0]	I	32 位输入当前指令，译码出选择信号
ALUC[31:0]	I	32 位输入 ALU 计算结果
DMRD[31:0]	I	32 位输入 DM 读出数据
pc4[31:0]	I	32 位输入当前 pc+4
HI[31:0]	I	32 位输入 HI
LO[31:0]	I	32 位输入 LO
RegData[31:0]	O	32 位输出存入寄存器数据

（19） MFCMP1D

功能描述	用于选择转发来的 rs 数据进入 CMP 和 E 级寄存器	
信号名	方向	描述
mfcmp1dSel[3:0]	I	4 位输入转发选择信号
RD1[31:0]	I	32 位输入当前位置指令从寄存器读取的数据
ALUC_M[31:0]	I	32 位输入来自 M 级的 ALU 计算结果
pc4_M[31:0]	I	32 位输入来自 M 级的 pc4
HI_M	I	32 位输入来自 M 级的 HI
LO_M	I	32 位输入来自 M 级的 LO
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果
DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
HI_W	I	32 位输入来自 W 级的 HI
LO_W	I	32 位输入来自 W 级的 LO
mfcmp1[31:0]	O	32 位输出选择出的当前指令的真正 rs 读取数据

（20） MFCMP2D

功能描述	用于选择转发来的 rt 数据进入 CMP 和 E 级寄存器	
信号名	方向	描述

mfcmp2dSel[3:0]	I	4 位输入转发选择信号
RD2[31:0]	I	32 位输入当前位置指令从寄存器读取的数据
ALUC_M[31:0]	I	32 位输入来自 M 级的 ALU 计算结果
pc4_M[31:0]	I	32 位输入来自 M 级的 pc4
HI_M	I	32 位输入来自 M 级的 HI
LO_M	I	32 位输入来自 M 级的 LO
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果
DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
HI_W	I	32 位输入来自 W 级的 HI
LO_W	I	32 位输入来自 W 级的 LO
mfcmp2[31:0]	O	32 位输出选择出的当前指令的真正 rt 读取数据

（21） MFALUAE

功能描述	用于选择转发来的真正寄存器值进入 ALU 作为 A	
信号名	方向	描述
mfaluaeSel[3:0]	I	4 位输入转发选择信号
V1_E[31:0]	I	32 位输入来自 E 级寄存器流水的 V1 值
ALUC_M[31:0]	I	32 位输入来自 M 级的 ALU 计算结果
pc4_M[31:0]	I	32 位输入来自 M 级的 pc4
HI_M	I	32 位输入来自 M 级的 HI
LO_M	I	32 位输入来自 M 级的 LO
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果
DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
HI_W	I	32 位输入来自 W 级的 HI
LO_W	I	32 位输入来自 W 级的 LO
mfalua[31:0]	O	32 位输出选择出的当前指令进入 ALU 的真正寄存器值

（22） MFALUBE

功能描述	用于选择转发来的真正寄存器值进入 ALU 的 B 值选择	
信号名	方向	描述
mfalubeSel[3:0]	I	4 位输入转发选择信号
V2_E[31:0]	I	32 位输入来自 E 级寄存器流水的 V2 值
ALUC_M[31:0]	I	32 位输入来自 M 级的 ALU 计算结果
pc4_M[31:0]	I	32 位输入来自 M 级的 pc4
HI_M	I	32 位输入来自 M 级的 HI
LO_M	I	32 位输入来自 M 级的 LO
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果
DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
HI_W	I	32 位输入来自 W 级的 HI
LO_W	I	32 位输入来自 W 级的 LO
mfalub[31:0]	O	32 位输出选择出的当前指令用于 ALUB 选择的真正寄存器值

（23） MFDM

功能描述	用于选择转发来的真正数据用于写入 DM	
信号名	方向	描述
mfdmSel[3:0]	I	4 位输入转发选择信号
V2_M[31:0]	I	32 位输入来自 M 级寄存器流水的 V2 值
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
HI_W	I	32 位输入来自 W 级的 HI
LO_W	I	32 位输入来自 W 级的 LO
mfdm[31:0]	O	32 位输出选择出的当前指令用于存入 DM 的真正数据

2. FORWARD_CONTROL

功能描述	立足 D 级，用于转发控制信号的生成	
信号名	方向	描述
instr_D[31:0]	I	32 位输入位于 D 级的指令
instr_E[31:0]	I	32 位输入位于 E 级的指令
instr_M[31:0]	I	32 位输入位于 M 级的指令
instr_W[31:0]	I	32 位输入位于 W 级的指令
A3_M[4:0]	I	5 位输入 M 级指令的写入寄存器地址
A3_W[4:0]	I	5 位输入 W 级指令的写入寄存器地址
mfcmp1dSel[3:0]	O	4 位输出 cmp1 转发数据选择信号
mfcmp2dSel[3:0]	O	4 位输出 cmp2 转发数据选择信号
mfaluaeSel[3:0]	O	4 位输出 ALUA 转发数据选择信号
mfalubeSel[3:0]	O	4 位输出 ALUB 转发数据选择信号
mfdmSel[3:0]	O	4 位输出 DM 转发数据选择信号

3. STOP_CONTROL

功能描述	放眼全局，用于暂停控制信号的生成	
信号名	方向	描述
instr_D[31:0]	I	32 位输入位于 D 级的指令
instr_E[31:0]	I	32 位输入位于 E 级的指令
instr_M[31:0]	I	32 位输入位于 M 级的指令
A3_E[4:0]	I	5 位输入 E 级指令的写入寄存器地址
A3_M[4:0]	I	5 位输入 M 级指令的写入寄存器地址
state_md	I	1 位输入表示乘法模块工作状态的信号
enPC	O	输出用于暂停的 PC 使能信号
enD	O	输出用于暂停的 D 级流水线寄存器使能信号
clrE	O	输出用于暂停的 E 级流水线寄存器清空信号

4. DECODE

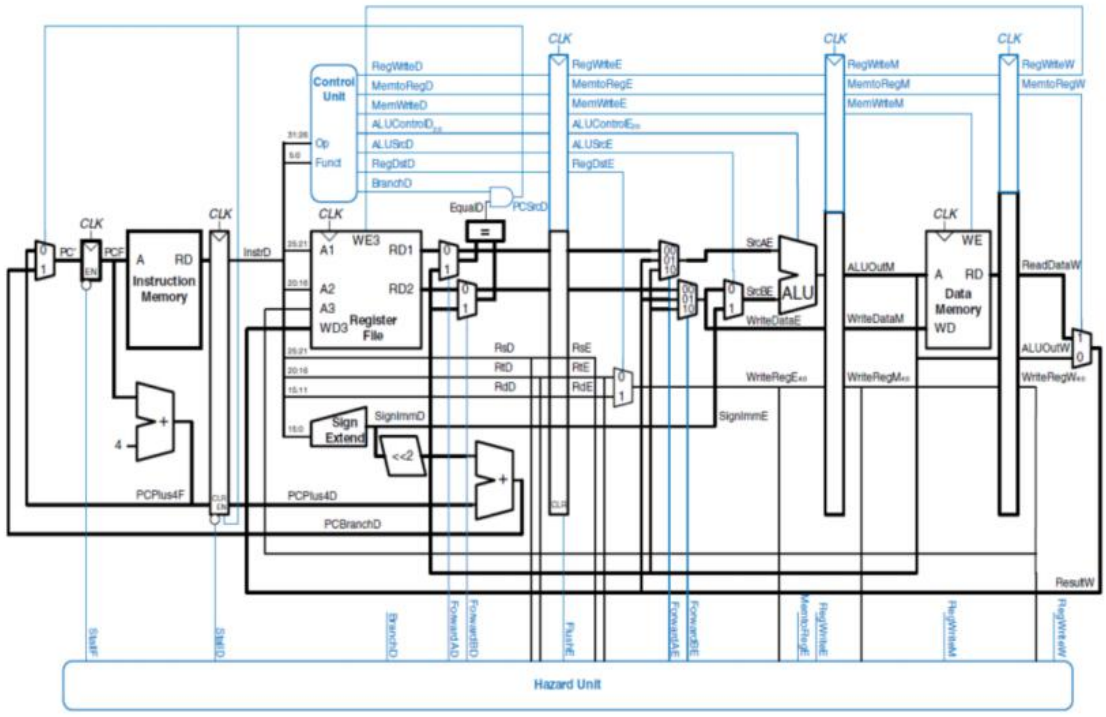
功能描述	用于译码产生各种控制信号

信号名	方向	描述
instr[31:0]	I	32 位输入指令信号
NPCOp[2:0]	O	3 位输出 npc 操作选择信号
EXTOp[3:0]	O	4 位输出 ext 操作选择信号
ALUOp[3:0]	O	4 位输出 ALU 操作选择信号
DMLOp[3:0]	O	4 位输出 DM 的 load 操作选择信号
DMSOp[3:0]	O	4 位输出 DM 的 store 操作选择信号
MDOp[3:0]	O	4 位输出 MD 的操作选择信号
start	O	1 位输出乘除模块工作开始信号
RegWrite	O	1 位输出寄存器写使能信号
MemWrite	O	1 位输出 DM 写使能信号
RegA3Sel[2:0]	O	3 位输出写入寄存器编号选择信号
RegDataSel[2:0]	O	3 位输出写入寄存器数据选择信号
AluBSel[2:0]	O	3 位输出 ALUB 选择信号
PcSel[1:0]	O	2 位输出 pc 次指令地址选择信号
Tuse_rs0	O	表示指令经 0 周期使用 rs 寄存器值
Tuse_rs1	O	表示指令经 1 周期使用 rs 寄存器值
Tuse_rt0	O	表示指令经 0 周期使用 rt 寄存器值
Tuse_rt1	O	表示指令经 1 周期使用 rt 寄存器值
Tuse_rt2	O	表示指令经 2 周期使用 rt 寄存器值
Tnew[2:0]	O	表示指令产生写入寄存器值的部件
ismd	O	1 位输出乘除法指令判断信号

（三）数据通路的综合

1. 所有指令的指令级别数据通路

		addu	subu	ori	lw	sw	beq	lui	j	jal	jr	nop	MUX	
PC		ADD4	ADD4	ADD4	ADD4	ADD4	ADD4|NPC	ADD4	NPC	NPC	RD1	ADD4	NPC	
IM		PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	
ADD4		PC	PC	PC	PC	PC	PC	PC	PC	PC		PC	PC	
D级	instr	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	
	pc4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4		ADD4	ADD4	
RF	A1	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D				instr[25:21]@D		instr[25:21]@D	
	A2	instr[20:16]@D	instr[20:16]@D			instr[20:16]@D	instr[20:16]@D						instr[20:16]@D	
EXT	imm			instr[25:0]@D	instr[25:0]@D	instr[25:0]@D		instr[25:0]@D					instr[25:0]@D	
NPC	pc4						pc4@D		pc4@D	pc4@D			pc4@D	
	imm						instr[25:0]@D		instr[25:0]@D	instr[25:0]@D			instr[25:0]@D	
	ra										RD1		RD1	
CMP	D1						RD1						MFCMP1D	
	D2						RD2						MFCMP2D	
E级	V1	RD1	RD1	RD1	RD1	RD1							MFCMP1D	
	V2	RD2	RD2			RD2							MFCMP2D	
	A1	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D							instr[25:21]@D	
	A2	instr[20:16]@D	instr[20:16]@D			instr[20:16]@D							instr[20:16]@D	
	A3	instr[15:11]@D	instr[15:11]@D	instr[20:16]@D	instr[20:16]@D			instr[20:16]@D		1f			MUX_A3	
	ext			EXT.ext	EXT.ext	EXT.ext		EXT.ext					EXT.ext	
	pc4									pc4@D			pc4@D	
ALU	A	V1@E	V1@E	V1@E	V1@E	V1@E		ext@E					MFALUAE	
	B	V2@E	V2@E	ext@E	ext@E	ext@E							MUX_ALUB	MFALUBE
	Y2					V2@E							MFALUBE	
M级	A2					A2@E							A2@E	
	ALUO	ALU.C	ALU.C	ALU.C	ALU.C	ALU.C		ALU.C					ALU.C	
	A3	A3@E	A3@E	A3@E	A3@E			A3@E		A3@E			A3@E	
	pc4									pc4@D			pc4@E	
DM	MemAddr				ALUO	ALUO@M							ALUO@M	
	MemData					V2@M							MFVDM	
W级	A3	A3@M	A3@M	A3@M	A3@M			A3@M		A3@M			A3@M	
	pc4									pc4@D			pc4	
	DMRD				DM[ALUO]								DM[ALUO]	
	ALUO	ALUO@M	ALUO@M	ALUO@M				ALUO@M					ALUO@M	
RF	A3	A3@M	A3@M	A3@M	A3@M			A3@W		A3@W			A3@M	
	RegData	ALUO@W	ALUO@W	ALUO@W	DMRD@W			ALUO@W		pc4@W			MUX_RegData	



（四）重要机制实现方法

1. 跳转

NPC 模块译码出 NPC 的计算控制信号，计算出 npc 后输出到 F 级 MUX_PcSel，MUX_PcSel 对来自 D 级寄存器的指令（即与 NPC 同指令）译码判断是否为跳转指令，若是则输出 npc 至 PC；反之输出当前 F 级 pc+4（来自 add4

部件的输出）至 PC。其中，对于 b 类指令，根据 CMP 模块的判断结果配合进行 NPC 模块的次指令地址选择。

2. 暂停

（部分指令的 Tuse 和 Tnew 表）

	rs	rt	功能部件	E	M	W
addu	1	1	ALU	1	0	0
subu	1	1	ALU	1	0	0
ori	1		ALU	1	0	0
lw	1		DM	2	1	0
sw	1	2				
beq	0	0				
lui			ALU	1	0	0
j						
jal			PC	0	0	0
jr	0					
nop						

	Tnew	E					M					W				
		HI	LO	ALU	DM	PC	HI	LO	ALU	DM	PC	HI	LO	ALU	DM	PC
rs	Tuse	1	1	1	2	0	1	1	0	1	0	1	1	0	0	0
	0	S	S	S	S	F	F	F	F	S	F	F	F	F	F	F
	1	F	F	F	S	F	F	F	F	F	F	F	F	F	F	F
	Tnew	E					M					W				
		HI	LO	ALU	DM	PC	HI	LO	ALU	DM	PC	HI	LO	ALU	DM	PC
rt	Tuse	1	1	1	2	0	1	1	0	1	0	1	1	0	0	0
	0	S	S	S	S	F	F	F	F	S	F	F	F	F	F	F
	1	F	F	F	S	F	F	F	F	F	F	F	F	F	F	F
	2	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

构建策略矩阵，当 Tnew>Tuse 时必须使用暂停，产生暂停信号，此时需要冻结 PC 的值，冻结 D 级流水线的值并清零 E 级流水线。暂停信号由 rs、rt 寄存器的暂停信号取并集，当 Tnew 与 Tuse 满足矩阵中暂停条件，且写入寄存器与读取寄存器相同时相应暂停信号置 1。对 D、E、M 级指令分别译码即可得到 Tuse 与 Tnew 值。

本实验中添加了乘除模块，根据现实情况，乘除模块工作时间较长（乘法为 5 个时钟周期，除法为 10 个时钟周期，此外还有开始工作的一个时钟周期），当乘除模块工作时，若新指令也需要使用乘除模块，则产生冲突，需要暂停，因此将乘除模块的 busy、start 信号连入暂停控制部件，当二者之并为 1 且 D 级指令为乘除指令时暂停。

此处需注意，当产生冲突的寄存器为 0 号寄存器时，不暂停。

3. 转发

沿用 Tnew 信号（此处似乎与教程和课件内容不甚相符），本设计在宏定义时不同 Tnew 信号反映该级目前指令产生写入寄存器值的部件，因此在转发控制中只需要根据该级处指令写入寄存器编号与 D 级指令读取寄存器编号相同、该级指令产生写入寄存器值的部件两个信号判断转发哪一数据。

对 M、W 级指令分别译码即可得到该两级产生的写入寄存器数据来源，从而明确转发的数据来源。

转发的“供给者”包括 M 级 ALU、PC4、HI、LO，W 级 ALU、PC4、DM、HI、LO，“需求者”包括 D 级的 cmp1，cmp2，E 级的 alua、alub，M 级的 dm。

此处需注意，如果写入寄存器为 0 号寄存器，不进行转发。

二、测试方案

（一）典型测试样例

Tuse_rs=0，Tnew_E=ALU，暂停	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ## beq $5, $2, label ## ori $10, $0, 5 ori $11, $0, 6 label: ori $12, $0, 0x302c ## jr $12 ## ori $1, $0, 9 lui $9, 0x2222 ori $12, $0, 326	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000 @00003014: $10 <= 00000005 @0000301c: $12 <= 0000302c @00003024: $ 1 <= 00000009 @0000302c: $12 <= 00000146
Tuse_rs=0，Tnew_E=DM，暂停	
ori $1, $0, 256	@00003000: $ 1 <= 00000100

ori $6, $0, 256
ori $5, $0, 4
sw $1, 4($5)
lw $10, 4($5) ##
beq $10, $6, label ##
nop
ori $20, $0, 2
label:
lui $4, 0x1234
	@00003004: $ 6 <= 00000100
	@00003008: $ 5 <= 00000004
	@0000300c: *00000008 <= 00000100
	@00003010: $10 <= 00000100
	@00003020: $ 4 <= 12340000

Tuse_rs=0，Tnew_E=PC（似乎这样的搭配只有延迟槽里跳转的情况）	
Tuse_rs=0，Tnew_M=ALU，转发	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ## ori $30, $0, 56 beq $5, $2, label ## ori $10, $0, 5 ori $11, $0, 6 label: ori $12, $0, 0x3034 ## ori $29, $0, 58 jr $12 ## ori $1, $0, 9 lui $9, 0x2222 ori $12, $0, 326	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000 @00003010: $30 <= 00000038 @00003018: $10 <= 00000005 @00003020: $12 <= 00003034 @00003024: $29 <= 0000003a @0000302c: $ 1 <= 00000009 @00003034: $12 <= 00000146
Tuse_rs=0，Tnew_M=DM，暂停	
ori $1, $0, 0x3038 ori $5, $0, 0x3038	@00003000: $ 1 <= 00003038 @00003004: $ 5 <= 00003038

ori $2, $0, 4
sw $1, -4($2)
lw $10, -4($2) ##
ori $3, $0, 123
beq $10, $5, label ##
nop
ori $25, $0, 256
label:
lw $6, -4($2) ##
lui $26, 0x1234
jr $6 ##
nop
ori $23, $0, 125
ori $24, $0, 156
	@00003008: $ 2 <= 00000004
	@0000300c: *00000000 <= 00003038
	@00003010: $10 <= 00003038
	@00003014: $ 3 <= 0000007b
	@00003024: $ 6 <= 00003038
	@00003028: $26 <= 12340000
	@00003038: $24 <= 0000009c

Tuse_rs=0，Tnew_M=PC，转发	
ori $5, $0, 0x3010 ori $1, $0, 256 jal label ## nop ori $2, $0, 255 label: beq $31, $5, label2 ## nop ori $6, $0, 247 label2: lui $9, 0x1234	@00003000: $ 5 <= 00003010 @00003004: $ 1 <= 00000100 @00003008: $31 <= 00003010 @00003020: $ 9 <= 12340000
Tuse_rs=0，Tnew_W=ALU，转发	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff

ori $4, $0, 1
addu $5, $3, $4 ##
ori $27, $0, 256
ori $22, $0, 355
beq $5, $2, label ##
ori $10, $0, 5
ori $11, $0, 6
label:
ori $12, $0, 0x303c ##
addu $23, $2, $4
subu $24, $23, $4
jr $12 ##
ori $1, $0, 9
lui $9, 0x2222
ori $12, $0, 326
	@00003008: $ 4 <= 00000001
	@0000300c: $ 5 <= 00003000
	@00003010: $27 <= 00000100
	@00003014: $22 <= 00000163
	@0000301c: $10 <= 00000005
	@00003024: $12 <= 0000303c
	@00003028: $23 <= 00003001
	@0000302c: $24 <= 00003000
	@00003034: $ 1 <= 00000009
	@0000303c: $12 <= 00000146

Tuse_rs=0，Tnew_W=DM，转发	

ori $1, $0, 0x3040
ori $5, $0, 0x3040
ori $2, $0, 4
sw $1, -4($2)
lw $10, -4($2) ##
ori $16, $0, 258
ori $3, $0, 123
beq $10, $5, label ##
nop
ori $25, $0, 256
label:
lw $6, -4($2) ##
lui $26, 0x1234
	@00003000: $ 1 <= 00003040
	@00003004: $ 5 <= 00003040
	@00003008: $ 2 <= 00000004
	@0000300c: *00000000 <= 00003040
	@00003010: $10 <= 00003040
	@00003014: $16 <= 00000102
	@00003018: $ 3 <= 0000007b
	@00003028: $ 6 <= 00003040
	@0000302c: $26 <= 12340000
	@00003030: $17 <= 0000017d
	@00003040: $24 <= 0000009c

addu $17, $16, $3 jr $6 ## nop ori $23, $0, 125 ori $24, $0, 156	
Tuse_rs=0，Tnew_W=PC，转发	
ori $5, $0, 0x3010 ori $1, $0, 256 jal label ## nop ori $2, $0, 255 label: ori $10, $0, 0x1234 beq $31, $5, label2 ## nop ori $6, $0, 247 label2: lui $9, 0x1234	@00003000: $ 5 <= 00003010 @00003004: $ 1 <= 00000100 @00003008: $31 <= 00003010 @00003014: $10 <= 00001234 @00003024: $ 9 <= 12340000
Tuse_rs=1，Tnew_E=ALU，转发	
ori $1, $0, 256 ori $2, $0, 255 ## addu $4, $2, $1 ##	@00003000: $ 1 <= 00000100 @00003004: $ 2 <= 000000ff @00003008: $ 4 <= 000001ff
Tuse_rs=1，Tnew_E=DM，暂停	

ori $1, $0, 4
ori $2, $0, 0x1234
sw $2, 4($1)
lw $3, 4($1) ##
addu $4, $3, $1 ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001234
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234
	@00003010: $ 4 <= 00001238

Tuse_rs=1，Tnew_E=PC，转发	

ori $1, $0, 154
ori $2, $0, 111
jal label ##
addu $3, $31, $1 ##
addu $4, $1, $2
label:
ori $5, $0, 147
	@00003000: $ 1 <= 0000009a
	@00003004: $ 2 <= 0000006f
	@00003008: $31 <= 00003010
	@0000300c: $ 3 <= 000030aa
	@00003014: $ 5 <= 00000093

Tuse_rs=1，Tnew_M=ALU，转发	
ori $1, $0, 256 ori $2, $0, 255 ## ori $3, $0, 289 addu $4, $2, $1 ##	@00003000: $ 1 <= 00000100 @00003004: $ 2 <= 000000ff @00003008: $ 3 <= 00000121 @0000300c: $ 4 <= 000001ff
Tuse_rs=1，Tnew_M=DM，转发	

ori $1, $0, 4
ori $2, $0, 0x1234
sw $2, 4($1)
lw $3, 4($1) ##
ori $25, $0, 156
addu $4, $3, $1 ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001234
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234
	@00003010: $25 <= 0000009c
	@00003014: $ 4 <= 00001238

Tuse_rs=1，Tnew_M=PC，转发	

ori $1, $0, 154
ori $2, $0, 111
jal label ##
addu $4, $1, $2
lui $8, 0x1235
label:
addu $3, $31, $1 ##
ori $5, $0, 147
	@00003000: $ 1 <= 0000009a
	@00003004: $ 2 <= 0000006f
	@00003008: $31 <= 00003010
	@0000300c: $ 4 <= 00000109
	@00003014: $ 3 <= 000030aa
	@00003018: $ 5 <= 00000093

Tuse_rs=1，Tnew_W=ALU，转发	

ori $1, $0, 256	@00003000: $ 1 <= 00000100
ori $2, $0, 255 ##	@00003004: $ 2 <= 000000ff
lui $5, 0x1256	@00003008: $ 5 <= 12560000
ori $3, $0, 289	@0000300c: $ 3 <= 00000121
addu $4, $2, $1 ##	@00003010: $ 4 <= 000001ff
Tuse_rs=1，Tnew_W=DM，转发	
ori $1, $0, 4	@00003000: $ 1 <= 00000004
ori $2, $0, 0x1234	@00003004: $ 2 <= 00001234
sw $2, 4($1)	@00003008: *00000008 <= 00001234
lw $3, 4($1) ##	@0000300c: $ 3 <= 00001234
ori $25, $0, 156	@00003010: $25 <= 0000009c
lui $26, 0x1475	@00003014: $26 <= 14750000
addu $4, $3, $1 ##	@00003018: $ 4 <= 00001238
Tuse_rs=1，Tnew_W=PC，转发	
ori $1, $0, 154	@00003000: $ 1 <= 0000009a
ori $2, $0, 111	@00003004: $ 2 <= 0000006f
jal label ##	@00003008: $31 <= 00003010
addu $4, $1, $2	@0000300c: $ 4 <= 00000109
lui $8, 0x1235	@00003014: $ 9 <= 00580000
label:	@00003018: $ 3 <= 000030aa
lui $9, 0x58	@0000301c: $ 5 <= 00000093
addu $3, $31, $1 ##	
ori $5, $0, 147	

Tuse_rt=0，Tnew_E=ALU，暂停	
ori $2, $0, 0x3000	@00003000: $ 2 <= 00003000
ori $3, $0, 0x2fff	@00003004: $ 3 <= 00002fff
ori $4, $0, 1	@00003008: $ 4 <= 00000001
addu $5, $3, $4 ##	@0000300c: $ 5 <= 00003000

beq $2, $5, label ## ori $10, $0, 5 ori $11, $0, 6 label: ori $12, $0, 0x302c	@00003014: $10 <= 00000005 @0000301c: $12 <= 0000302c
Tuse_rt=0，Tnew_E=ALU，暂停	

ori $1, $0, 256
ori $6, $0, 256
ori $5, $0, 4
sw $1, 4($5)
lw $10, 4($5) ##
beq $6, $10, label ##
nop
ori $20, $0, 2
label:
lui $4, 0x1234
	@00003000: $ 1 <= 00000100
	@00003004: $ 6 <= 00000100
	@00003008: $ 5 <= 00000004
	@0000300c: *00000008 <= 00000100
	@00003010: $10 <= 00000100
	@00003020: $ 4 <= 12340000

Tuse_rt=0，Tnew_E=PC（似乎这样的搭配只有延迟槽里跳转的情况）	
Tuse_rt=0，Tnew_M=ALU，转发	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ## ori $30, $0, 56 beq $2, $5, label ## ori $10, $0, 5 ori $11, $0, 6 label: ori $12, $0, 0x3034	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000 @00003010: $30 <= 00000038 @00003018: $10 <= 00000005 @00003020: $12 <= 00003034
Tuse_rt=0，Tnew_M=DM，暂停	

ori $1, $0, 0x3038
ori $5, $0, 0x3038
ori $2, $0, 4
sw $1, -4($2)
lw $10, -4($2) ##
ori $3, $0, 123
beq $5, $10, label ##
nop
ori $25, $0, 256
label:
ori $27, $0, 145
	@00003000: $ 1 <= 00003038
	@00003004: $ 5 <= 00003038
	@00003008: $ 2 <= 00000004
	@0000300c: *00000000 <= 00003038
	@00003010: $10 <= 00003038
	@00003014: $ 3 <= 0000007b
	@00003024: $27 <= 00000091

Tuse_rt=0，Tnew_M=PC，转发	
ori $5, $0, 0x3011 ori $1, $0, 256 jal label ## nop ori $2, $0, 255 label: beq $5, $31, label2 ## nop ori $6, $0, 247 label2: lui $9, 0x1234	@00003000: $ 5 <= 00003011 @00003004: $ 1 <= 00000100 @00003008: $31 <= 00003010 @0000301c: $ 6 <= 000000f7 @00003020: $ 9 <= 12340000
Tuse_rt=0，Tnew_W=ALU，转发	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ## ori $27, $0, 256	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000 @00003010: $27 <= 00000100

ori $22, $0, 355 beq $2, $5, label ## ori $10, $0, 5 ori $11, $0, 6 label: addu $23, $2, $4	@00003014: $22 <= 00000163 @0000301c: $10 <= 00000005 @00003024: $23 <= 00003001
Tuse_rt=0，Tnew_W=DM，转发	

ori $1, $0, 0x3040
ori $5, $0, 0x3040
ori $2, $0, 4
sw $1, -4($2)
lw $10, -4($2) ##
ori $16, $0, 258
ori $3, $0, 123
beq $5, $10, label ##
nop
ori $25, $0, 256
label:
lw $6, -4($2)
	@00003000: $ 1 <= 00003040
	@00003004: $ 5 <= 00003040
	@00003008: $ 2 <= 00000004
	@0000300c: *00000000 <= 00003040
	@00003010: $10 <= 00003040
	@00003014: $16 <= 00000102
	@00003018: $ 3 <= 0000007b
	@00003028: $ 6 <= 00003040

Tuse_rt=0，Tnew_W=PC，转发	
ori $5, $0, 0x3010 ori $1, $0, 256 jal label ## nop ori $2, $0, 255 label: ori $10, $0, 0x1234 beq $5, $31, label2 ## nop	@00003000: $ 5 <= 00003010 @00003004: $ 1 <= 00000100 @00003008: $31 <= 00003010 @00003014: $10 <= 00001234 @00003024: $ 9 <= 12340000

ori $6, $0, 247 label2: lui $9, 0x1234	
Tuse_rt=1，Tnew_E=ALU，转发	
ori $1, $0, 256 ori $2, $0, 255 ## addu $4, $1, $2 ##	@00003000: $ 1 <= 00000100 @00003004: $ 2 <= 000000ff @00003008: $ 4 <= 000001ff
Tuse_rt=1，Tnew_E=DM，暂停	

ori $1, $0, 4
ori $2, $0, 0x1234
sw $2, 4($1)
lw $3, 4($1) ##
addu $4, $1, $3 ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001234
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234
	@00003010: $ 4 <= 00001238

Tuse_rt=1，Tnew_E=PC，转发	

ori $1, $0, 154
ori $2, $0, 111
jal label ##
addu $3, $1, $31 ##
addu $4, $1, $2
label:
ori $5, $0, 147
	@00003000: $ 1 <= 0000009a
	@00003004: $ 2 <= 0000006f
	@00003008: $31 <= 00003010
	@0000300c: $ 3 <= 000030aa
	@00003014: $ 5 <= 00000093

Tuse_rt=1，Tnew_M=ALU，转发	
ori $1, $0, 256 ori $2, $0, 255 ## ori $3, $0, 289 addu $4, $1, $2 ##	@00003000: $ 1 <= 00000100 @00003004: $ 2 <= 000000ff @00003008: $ 3 <= 00000121 @0000300c: $ 4 <= 000001ff
Tuse_rt=1，Tnew_M=DM，转发	
ori $1, $0, 4 ori $2, $0, 0x1234	@00003000: $ 1 <= 00000004 @00003004: $ 2 <= 00001234

sw $2, 4($1)
lw $3, 4($1) ##
ori $25, $0, 156
addu $4, $1, $3 ##
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234
	@00003010: $25 <= 0000009c
	@00003014: $ 4 <= 00001238

Tuse_rt=1，Tnew_M=PC，转发	

ori $1, $0, 154
ori $2, $0, 111
jal label ##
addu $4, $1, $2
lui $8, 0x1235
label:
addu $3, $31, $1 ##
ori $5, $0, 147
	@00003000: $ 1 <= 0000009a
	@00003004: $ 2 <= 0000006f
	@00003008: $31 <= 00003010
	@0000300c: $ 4 <= 00000109
	@00003014: $ 3 <= 000030aa
	@00003018: $ 5 <= 00000093

Tuse_rt=1，Tnew_W=ALU，转发	
ori $1, $0, 256 ori $2, $0, 255 ## lui $5, 0x1256 ori $3, $0, 289 addu $4, $1, $2 ##	@00003000: $ 1 <= 00000100 @00003004: $ 2 <= 000000ff @00003008: $ 5 <= 12560000 @0000300c: $ 3 <= 00000121 @00003010: $ 4 <= 000001ff
Tuse_rt=1，Tnew_W=DM，转发	

ori $1, $0, 4
ori $2, $0, 0x1234
sw $2, 4($1)
lw $3, 4($1) ##
ori $25, $0, 156
lui $26, 0x1475
addu $4, $1, $3 ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001234
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234
	@00003010: $25 <= 0000009c
	@00003014: $26 <= 14750000
	@00003018: $ 4 <= 00001238

Tuse_rt=1，Tnew_W=PC，转发	
ori $1, $0, 154	@00003000: $ 1 <= 0000009a

ori $2, $0, 111
jal label ##
addu $4, $1, $2
lui $8, 0x1235
label:
lui $9, 0x58
addu $3, $1, $31 ##
ori $5, $0, 147
	@00003004: $ 2 <= 0000006f
	@00003008: $31 <= 00003010
	@0000300c: $ 4 <= 00000109
	@00003014: $ 9 <= 00580000
	@00003018: $ 3 <= 000030aa
	@0000301c: $ 5 <= 00000093

Tuse_rt=2，Tnew_E=ALU，转发	

ori $1, $0, 4
ori $5, $0, 256
addu $3, $1, $1 ##
sw $3, 4($5) ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 5 <= 00000100
	@00003008: $ 3 <= 00000008
	@0000300c: *00000104 <= 00000008

Tuse_rt=2，Tnew_E=DM，转发	

ori $1, $0, 4
ori $2, $0, 0x1243
sw $2, -4($1)
lw $3, -4($1) ##
sw $3, 8($1) ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001243
	@00003008: *00000000 <= 00001243
	@0000300c: $ 3 <= 00001243
	@00003010: *0000000c <= 00001243

Tuse_rt=2，Tnew_E=PC，转发	

ori $1, $0, 4
jal label ##
sw $31, 4($1) ##
addu $3, $1, $1
label:
ori $5, $0, 156
	@00003000: $ 1 <= 00000004
	@00003004: $31 <= 0000300c
	@00003008: *00000008 <= 0000300c
	@00003010: $ 5 <= 0000009c

Tuse_rt=2，Tnew_M=ALU，转发	
ori $1, $0, 4 ori $5, $0, 256	@00003000: $ 1 <= 00000004 @00003004: $ 5 <= 00000100

addu $3, $1, $1 ##	@00003008: $ 3 <= 00000008
ori $4, $0, 28	@0000300c: $ 4 <= 0000001c
sw $3, 4($5) ##	@00003010: *00000104 <= 00000008
Tuse_rt=2，Tnew_M=DM，转发	
ori $1, $0, 4	@00003000: $ 1 <= 00000004
ori $2, $0, 0x1243	@00003004: $ 2 <= 00001243
sw $2, -4($1)	@00003008: *00000000 <= 00001243
lw $3, -4($1) ##	@0000300c: $ 3 <= 00001243
ori $9, $0, 156	@00003010: $ 9 <= 0000009c
sw $3, 8($1) ##	@00003014: *0000000c <= 00001243
Tuse_rt=2，Tnew_M=PC，转发	
ori $1, $0, 4	@00003000: $ 1 <= 00000004
jal label ##	@00003004: $31 <= 0000300c
nop	@00003010: *00000008 <= 0000300c
addu $3, $1, $1	@00003014: $ 5 <= 0000009c
label:	
sw $31, 4($1) ##	
ori $5, $0, 156	
Tuse_rt=2，Tnew_W=ALU，转发	
ori $1, $0, 4	@00003000: $ 1 <= 00000004
ori $5, $0, 256	@00003004: $ 5 <= 00000100
addu $3, $1, $1 ##	@00003008: $ 3 <= 00000008
ori $4, $0, 28	@0000300c: $ 4 <= 0000001c
addu $7, $1, $5	@00003010: $ 7 <= 00000104
sw $3, 4($5) ##	@00003014: *00000104 <= 00000008
Tuse_rt=2，Tnew_W=DM，转发	
ori $1, $0, 4	@00003000: $ 1 <= 00000004
ori $2, $0, 0x1243	@00003004: $ 2 <= 00001243
sw $2, -4($1)	@00003008: *00000000 <= 00001243

lw $3, -4($1) ##	@0000300c: $ 3 <= 00001243
ori $9, $0, 156	@00003010: $ 9 <= 0000009c
addu $5, $1, $2	@00003014: $ 5 <= 00001247
sw $3, 8($1) ##	@00003018: *0000000c <= 00001243
Tuse_rt=2，Tnew_W=PC，转发	
ori $1, $0, 4	@00003000: $ 1 <= 00000004
jal label ##	@00003004: $31 <= 0000300c
nop	@00003010: $ 8 <= 00000008
addu $3, $1, $1	@00003014: *00000008 <= 0000300c
label:	@00003018: $ 5 <= 0000009c
addu $8, $1, $1	
sw $31, 4($1) ##	
ori $5, $0, 156	

乘除模块测试：

ori $2, $0, 6	@00003000: $ 2 <= 00000006
ori $3, $0, 2	@00003004: $ 3 <= 00000002
ori $4, $0, 3	@00003008: $ 4 <= 00000003
ori $31, $0, 1024	@0000300c: $31 <= 00000400
mult $3, $4	@00003014: $ 5 <= 00000006
mflo $5 ##	@0000301c: $10 <= 00000005
beq $5, $2, label ##	@00003024: $11 <= 00000c0f
ori $10, $0, 5	@00003028: $15 <= 00000004
ori $11, $0, 6	@00003030: $12 <= 0000303c
label:	@00003038: $ 1 <= 00000009
ori $11, $0, 0xc0f	@0000303c: $ 9 <= 22220000
ori $15, $0, 4	@00003040: $12 <= 00000146
mult $11, $15	@00003044: $13 <= 00000004

mflo $12 ##	@00003048: $13 <= 00000000
jr $12 ##	@0000304c: *00000004 <= 00000006
ori $1, $0, 9	@0000305c: $22 <= 00000000
lui $9, 0x2222	@00003060: $23 <= 00000003
ori $12, $0, 326	@00003064: $14 <= 00000100
ori $13, $0, 4	@00003068: $15 <= ffff0000
mfhi $13	@00003070: $26 <= ffffffff
sw $2, 4($13)	@00003074: $27 <= ff000000
mthi $2	@00003080: $27 <= ff000000
mtlo $3	@0000308c: $26 <= 22220000
div $2, $3	@00003090: $27 <= 00ffff00
mfhi $22	@0000309c: $26 <= 00000000
mflo $23	@000030a0: $27 <= 00000400
ori $14, $0, 256	@000030a4: *000003fc <= 00000400
lui $15, 0xffff	@000030b0: $26 <= 00000000
mult $15, $14	@000030b4: $27 <= 00000400
mfhi $26	@000030b8: $30 <= 00000400
mflo $27	
multu $15, $14	
mthi $26	
mflo $27	
divu $15, $14	
mthi $9	
mfhi $26	
mflo $27	
div $15, $14	
mtlo $31	
mfhi $26	
mflo $27	

sw $31, -4($31)
div $15, $14
mtlo $31
mfhi $26
mflo $27
lw $30, -4($31)	

DM 新指令测试：

ori $30, $0, 256	@00003000: $30 <= 00000100
ori $1, $0, 0x1234	@00003004: $ 1 <= 00001234
lui $2, 0x5678	@00003008: $ 2 <= 56780000
addu $3, $1, $2	@0000300c: $ 3 <= 56781234
ori $4, $0, 0x98a1	@00003010: $ 4 <= 000098a1
lui $5, 0xb2c3	@00003014: $ 5 <= b2c30000
addu $6, $4, $5	@00003018: $ 6 <= b2c398a1
ori $15, $0, 4	@0000301c: $15 <= 00000004
sw $3, 4($15)	@00003020: *00000008 <= 56781234
sw $6, 8($15)	@00003024: *0000000c <= b2c398a1
lw $21, 4($15)	@00003028: $21 <= 56781234
addu $31, $21, $30	@0000302c: $31 <= 56781334
lbu $21, 4($15)	@00003030: $21 <= 00000034
addu $31, $21, $30	@00003034: $31 <= 00000134
lbu $21, 5($15)	@00003038: $21 <= 00000012
addu $31, $21, $30	@0000303c: $31 <= 00000112
lbu $21, 6($15)	@00003040: $21 <= 00000078
addu $31, $21, $30	@00003044: $31 <= 00000178
lbu $21, 8($15)	@00003048: $21 <= 000000a1
addu $31, $21, $30	@0000304c: $31 <= 000001a1

lbu $21, 9($15)	@00003050: $21 <= 00000098
addu $31, $21, $30	@00003054: $31 <= 00000198
lbu $21, 6($15)	@00003058: $21 <= 00000078
addu $31, $21, $30	@0000305c: $31 <= 00000178
lbu $21, 8($15)	@00003060: $21 <= 000000a1
addu $31, $21, $30	@00003064: $31 <= 000001a1
lbu $21, 10($15)	@00003068: $21 <= 000000c3
addu $31, $21, $30	@0000306c: $31 <= 000001c3
sb $6, 0($15)	@00003070: *00000004 <= 000000a1
sb $6, 1($15)	@00003074: *00000004 <= 0000a1a1
sb $6, 2($15)	@00003078: *00000004 <= 00a1a1a1
sb $6, 3($15)	@0000307c: *00000004 <= a1a1a1a1
sh $6, 12($15)	@00003080: *00000010 <= 000098a1
sh $6, 14($15)	@00003084: *00000010 <= 98a198a1
lw $25, 0($15)	@00003088: $25 <= a1a1a1a1
lw $25, 4($15)	@0000308c: $25 <= 56781234
lw $25, 8($15)	@00003090: $25 <= b2c398a1
lw $25, 12($15)	@00003094: $25 <= 98a198a1

三、思考题

（一）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？

答：

一方面，根据 MIPS 指令集，ALU 的工作模式是直接计算并立即将计算结果传出使写入某一寄存器，而乘除法 mult、multu、div、divu 并非在计算结束后直接将结果写入寄存器，而是需要通过新指令 mflo、mfhi 将最近产生的结果写入寄存器，这与 ALU 的工作步骤不符，因此无法直接整合进 ALU。另一方面，在实际工程中，乘除法并不是 verilog 中乘除符号一样的简单计算，其计算过程

较长（本实验中为 5 周期或 10 周期），而 ALU 中其他计算均为 1 周期结束，因此二者的时序不符，无法将乘除法部件加入到 ALU 中。

Verilog 建模的流水线 CPU 与 MIPS 的 CPU 构成基本相符，HI、LO 两个独立寄存器也正是对 MIPS 的设置的遵从。对于乘法，两 32 位数相乘其结果位数最高可能达到 64 位，因此计算需要两个 32 位变量分别存储高 32 位和低 32 位；对于除法，将产生商和余数两个 32 位数，同样需要两个 32 位变量分别存储。与此同时，根据 MIPS 指令的规则，一条指令最多只能存入一个寄存器，因此在乘除模块中就需要两个独立的寄存器存储计算产生的两个结果，之后再根据需要决定是否、什么时候将这两个独立寄存器中的数据存入到 GRF 中，这样也更加快速，无需再次计算。

（二）参照你对延迟槽的理解，试解释“乘除槽”。

答：

乘除法在乘除模块中开始进行后，需要 5 或 10 周期才能产生结果，在这个时间中，后面的指令可以继续进入流水线，此时需要在 D 级对指令进行判断，如果为乘除指令，则因乘除模块的占用而暂停，若为其他指令，其操作与乘除不相干扰，可以继续执行，由此可以提升 CPU 的效率。

（三）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）

答：

由 c 语言中的字符串可知，一个英文字母占用一个字节，因此在处理字符串时需要以字节为单位进行处理，尤其是当处理数量不为四的倍数时按字节访问在性能上更具优势。

（四）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？

答：

延用 P5 的思考方法，将指令分为 8 类：

cal_r	add、addu、andd、norr、orr、slt、sltu、sllv、srav、srlv、sub、subu、xorr
cal_i	addi、addiu、andi、lui、ori、slti、sltiu、xori
load	lb、lbu、lh、lhu、lw
store	sb、sh、sw
bType	beq、bgez、bgtz、blez、bltz、bne
jType	jr、jalr、j、jal
shift	sll、sra、srl
lsmd（乘除模块）	div、divu、mfhi、mflo、mthi、mtlo、mult、multu

根据策略矩阵与指令分类构造测试样例，重点测试新加入的乘除模块。

（四）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？

答：

首先将数十条代码分为了八类（详见上一个问题），相同类中指令的数据通路基本相同，以类名变量代替这十几个变量，进而加入到控制信号的表达式中，因此在添加指令时只需要将新指令添加到其所属类的表达式中即可，有效减少工作量。

对于 b 类指令，不同跳转的判断内容不同，我在 CMP 模块中进行了译码，直接将六个 b 类指令翻译出来，针对不同指令进行相应的判断，判断结果全部用一个变量 zero 表示：

```
assign zero = (instr[31:26] == 6'b000100 && D1 == D2) ? 1 :
              (instr[31:26] == 6'b000101 && D1 != D2) ? 1 :
              (instr[31:26] == 6'b000110 && $signed(D1) <= 0) ? 1 :
              (instr[31:26] == 6'b000001 && instr[20:16] == 5'b00000 && $signed(D1) < 0) ? 1 :
              (instr[31:26] == 6'b000111 && $signed(D1) > 0) ? 1 :
              (instr[31:26] == 6'b000001 && instr[20:16] == 5'b00001 && $signed(D1) >= 0) ? 1 : 0;
```

这样也可以避免 NPC 模块中控制信号的增加导致的混乱。

对于有共性的内容，在一个模块中可以先将共同的内容表示出来，之后根据不同的操作信号进行进一步的处理，比如 DM 模块中各指令都需要数据存储器中相应整地址读取的数据，因此可以将其先赋给变量 readword，之后根据不同指令要求进行进一步的截取或拼接。

将指令分类，各类别的 AT 基本完全相同，因此可以构造简单的策略矩阵，

这也使得本 CPU 的冲突部分代码量很少，无需逐条考虑冲突，而且在构造测试样例时也可以根据指令类别构造出一整类的覆盖性测试样例，减轻思考负担。

各类别指令的数据通路基本相同，因此在译码时也可直接将一整类添加至操作信号表达式中，但与此同时，可以根据类别内部指令的差异性构造操作信号，我认为将每个指令单独配置操作信号可以有效降低思考量与错误率（或许增加一些代码量也是降低复杂性的一种方法）。