计算机组成原理 P7 实验报告

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 实现的包含中断与异常处理的流水线 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、ERET、MFC0、MTC0}。

在顶层模块 mips.v 下分 CPU、BRIDGE、Timer0、Timer1 四个部分，其中 Timer0 与 Timer1 负责提供外部中断，由桥与 CPU 沟通。

为满足中断与异常的处理，CPU 中增加了关键模块 CP0，置于数据通路的 M 级，内设 SR、CAUSE、EPC、PRId 四个寄存器。PC 模块增添判断 pc 引发的取指异常的功能，D 级增设 ckeckD 模块用于判断未知指令码的异常，E 级增设 checkE 模块，用于判断 ALU 加减法运算异常，M 级增设 checkM 模块，用于判断 DM 存取异常。

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

5. CP0

功能描述	用于存储或读取异常、中断产生的信息	
信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
instr_M[31:0]	I	32 位输入 M 级指令
instr_W[31:0]	I	32 位输入 W 级指令
CP0Data[31:0]	I	32 位输入存入 CP0 的值
HWInt[7:2]	I	6 位输入外部中断信号
exccode[6:2]	I	5 位输入异常种类编码
backPc_D[31:0]	I	32 位输入 D 级指令对应的返回 pc
backPc_E[31:0]	I	32 位输入 E 级指令对应的返回 pc
backPc_M[31:0]	I	32 位输入 M 级指令对应的返回 pc
afterJump_D	I	1 位输入判断 D 级指令是否为延迟槽指令
afterJump_E	I	1 位输入判断 E 级指令是否为延迟槽指令
afterJump_M	I	1 位输入判断 M 级指令是否为延迟槽指令
exccode_Dn[6:2]	I	5 位输入 D 级指令对应的异常编码
exccode_En[6:2]	I	5 位输入 E 级指令对应的异常编码
exccode_Mn[6:2]	I	5 位输入 M 级指令对应的异常编码
macroPc[31:0]	O	32 位输出宏观 pc
CP0RD[31:0]	O	32 位输出 CP0 中寄存器读取的值
epc[31:2]	O	30 位输出 epc 的高 30 位
exc_int	O	1 位输出异常和中断总信号

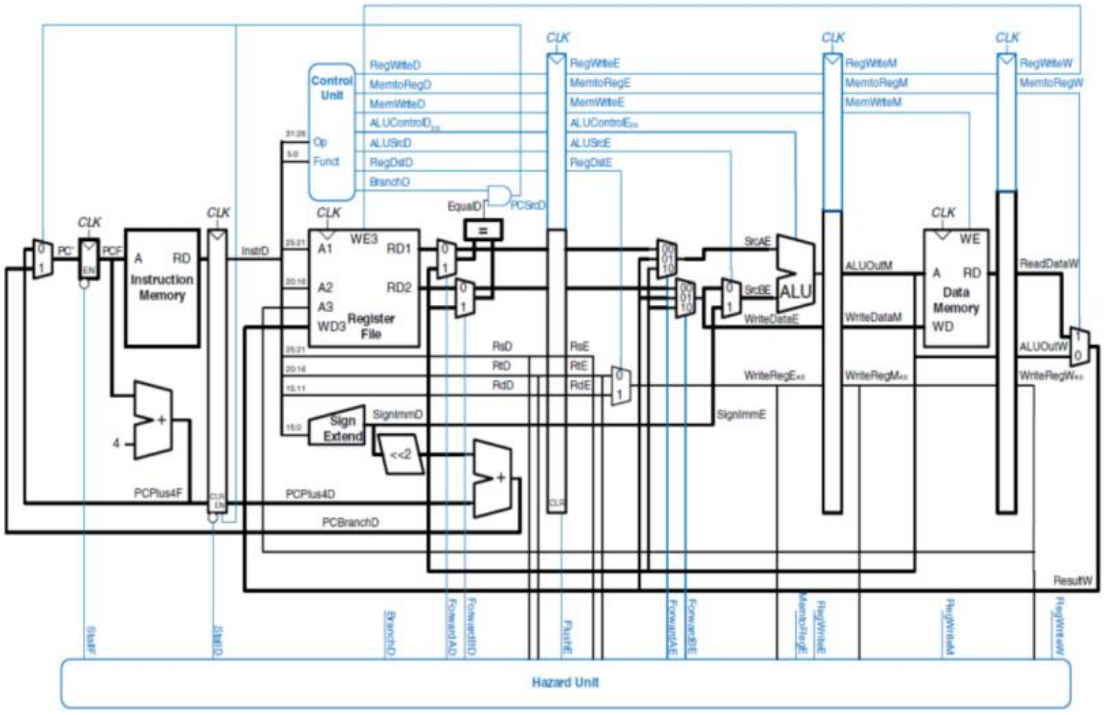
6. BRIDGE

功能描述	用于连接 CPU 与外部设备	
信号名	方向	描述
interrupt	I	1 位输入外部中断信号
PrAddr[31:2]	I	30 位输入外部设备地址，按字访问

PrWD[31:0]	I	32 位输入存入外部设备的数据
PrWrite	I	1 位输入外部设备写使能
timer0_RD[31:0]	I	32 位输入 Timer0 读出数据
Timer1_RD[31:0]	I	32 位输入 Timer1 读出数据
timer0_IRQ	I	1 位输入 Timer0 中断请求
Timer1_IRQ	I	1 位输入 Timer1 中断请求
PrRD[31:0]	O	32 位输出外部设备读取数据
HWInt[7:2]	O	6 位输出总外部中断记录
timer0Write	O	1 位输出 Timer0 写使能
Timer1Write	O	1 位输出 Timer1 写使能
device_Addr[31:2]	O	30 位输出外部设备写入地址
device_WD[31:0]	O	32 位输出外部设备写入数据

（三）数据通路的综合

1. 所有指令的指令级别数据通路



（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

Mips 代码	Verilog 运行结果
.text ori $24, $0, 0x1401 mtc0 $24, $12 ori $1, $0, 11 ori $2, $0, 7 ori $3,$0, 0x7f00 ori $4, $0, 0x7f04 ori $5, $0, 0x7f10	@00003000: $24 <= 00001401 @00003008: $ 1 <= 0000000b @0000300c: $ 2 <= 00000007 @00003010: $ 3 <= 00007f00 @00003014: $ 4 <= 00007f04 @00003018: $ 5 <= 00007f10 @0000301c: $ 6 <= 00007f14 @00003020: $ 7 <= 00000006

ori $6, $0, 0x7f14	@00004180: $31 <= 00003034
ori $7, $0, 6	@00004184: $30 <= 00003034
sw $1, 0($3)	@0000418c: $30 <= 00003040
sw $2, 0($4)	@000041e4: $ 6 <= 00007f00
sw $1, 0($5)	@00003038: $10 <= 7fff0000
sw $7, 0($6)	@0000303c: $11 <= 7fff0000
sw $7, 4($6)	@00003040: $31 <= 00003048
	@00003044: $12 <= fffe0000
lui $10, 0x7fff	@0000304c: $15 <= 00005000
lui $11, 0x7fff	@00004180: $31 <= 00003050
jal label	@00004184: $30 <= 00003034
addu $12, $10, $11	@0000418c: $30 <= 00003040
addu $13, $12, $12	@00004198: $30 <= 00003048
	@000041a4: $30 <= 00005000
label:	@000041b0: $30 <= 0000305c
ori $15, $0, 0x5000	@000041bc: $30 <= 00003068
ori $16, $0, 15	@000041c8: $30 <= 00003000
jr $15	@000041d4: $30 <= 0000309c
nop	@00003050: $16 <= 0000000f
sh $16, 0($4)	@00004180: $31 <= 00005000
ori $18, $0, 5	@00004184: $30 <= 00003034
lui $17, 0x8000	@0000418c: $30 <= 00003040
jal note	@00004198: $30 <= 00003048
subu $19, $17, $18	@000041a4: $30 <= 00005000
ori $20, $0, 15	@000041fc: $15 <= 0000305c
	@00004180: $31 <= 0000305c
note:	@00004184: $30 <= 00003034
nop	@0000418c: $30 <= 00003040
nop	@00004198: $30 <= 00003048

nop	@000041a4: $30 <= 00005000
nop	@000041b0: $30 <= 0000305c
nop	@00004208: $ 4 <= 00000006
nop	@0000305c: *00000004 <=
nop	000f0000
ori $20, $0, 0x3002	@00003060: $18 <= 00000005
jr $20	@00003064: $17 <= 80000000
nop	@00003068: $31 <= 00003070
addu $10, $11, $12##change to ffffffff	@0000306c: $19 <= 7ffffffb
ori $18, $0, 145	@00004180: $31 <= 00003074
ori $27, $0, 265	@00004184: $30 <= 00003034
mfc0, $29, $14	@0000418c: $30 <= 00003040
end:	@00004198: $30 <= 00003048
beq $0, $0,end	@000041a4: $30 <= 00005000
nop	@000041b0: $30 <= 0000305c
	@000041bc: $30 <= 00003068
.ktext 0x4180	@000041c8: $30 <= 00003000
mfc0 $31, $14	@000041d4: $30 <= 0000309c
	@00004180: $31 <= 00003090
ori $30, $0, 0x3034	@00004184: $30 <= 00003034
beq $31, $30, special1	@0000418c: $30 <= 00003040
	@00004198: $30 <= 00003048
ori $30, $0, 0x3040	@000041a4: $30 <= 00005000
beq $31, $30, special2	@000041b0: $30 <= 0000305c
nop	@000041bc: $30 <= 00003068
	@000041c8: $30 <= 00003000
ori $30, $0, 0x3048	@000041d4: $30 <= 0000309c
beq $31, $30, special3	@00003090: $20 <= 00003002
nop	@00004180: $31 <= 00003000

	@00004184: $30 <= 00003034
ori $30, $0, 0x5000	@0000418c: $30 <= 00003040
beq $30, $31, special4	@00004198: $30 <= 00003048
nop	@000041a4: $30 <= 00005000
	@000041b0: $30 <= 0000305c
ori $30, $0, 0x305c	@000041bc: $30 <= 00003068
beq $31, $30, special5	@000041c8: $30 <= 00003000
nop	@00004218: $20 <= 0000309c
	@00004180: $31 <= 0000309c
ori $30, $0, 0x3068	@00004184: $30 <= 00003034
beq $31, $30, special6	@0000418c: $30 <= 00003040
nop	@00004198: $30 <= 00003048
	@000041a4: $30 <= 00005000
ori $30, $0, 0x3000	@000041b0: $30 <= 0000305c
beq $31, $30, special7	@000041bc: $30 <= 00003068
nop	@000041c8: $30 <= 00003000
	@000041d4: $30 <= 0000309c
ori $30, $0, 0x309c	@00004224: $31 <= 000030a4
beq $31, $30 special8	@000030a4: $27 <= 00000109
nop	@000030a8: $29 <= 000030a4
eret	
special1:	
ori $6, $0, 0x7f00	
eret	
special2:	
ori $11, $0, 0	
eret	

```
special3:
ori $12, $0, 20
eret

special4:
ori $15, $0, 0x305c
mtc0 $15, $14
eret

special5:
ori $4, $0, 6
eret

special6:
ori $17, $0, 5
eret

special7:
ori $20, $0, 0x309c
mtc0 $20, $14
eret

special8:
ori $31, $0, 0x30a4
mtc0 $31, $14
eret
```

三、思考题

（一）我们计组课程一本参考书目标题中有“硬件/软件接口”接口字样，那么到底什么是“硬件/软件接口”？（Tips：什么是接口？和我们到现在为止所学的有什么联系？

答：

硬件/软件接口指将二者联系起来的部分，比如本实验中 CPU 正常运行的用户态和进入异常的内核态之间的连接部分，我认为可以包括将什么认定为异常、中断的这种法则。

百度：接口（硬件类接口）是指同一计算机不同功能层之间的通信规则称为接口。接口（软件类接口）是指对协定进行定义的引用类型。其他类型实现接口，以保证它们支持某些操作。接口指定必须由类提供的成员或实现它的其他接口。与类相似，接口可以包含方法、属性、索引器和事件作为成员。

（二）在我们设计的流水线中，DM 处于 CPU 内部，请你考虑现代计算机中它的位置应该在何处。

答：

现代计算机的 DM（主存）位于 CPU 外部，即与其他外接部件并列。我认为这是因为 CPU 主要负责运算，其需要较小的容量，较大的速度；而 DM 主要负责存储，其需要较大的空间，将 DM 置于 CPU 内部可能拖慢 CPU 速度，而且使内部接线更加复杂。

（三）BE 部件对所有的外设都是必要的吗？

答：

我认为不是。至少在我目前的 CPU 设计中并没有使用 BE 信号。我猜想这是因为外部仅有的 TImer 部件仅支持整字的存取，但另一方面，正如 P6 中我的 CPU 没有设置用于截取位数的扩展 DM 模块，我认为在理想情况下也可以通过译码出具体指令的方式进行按特定位数存取的操作（当然，现实生活中必然不能这么干，太浪费资源了）。

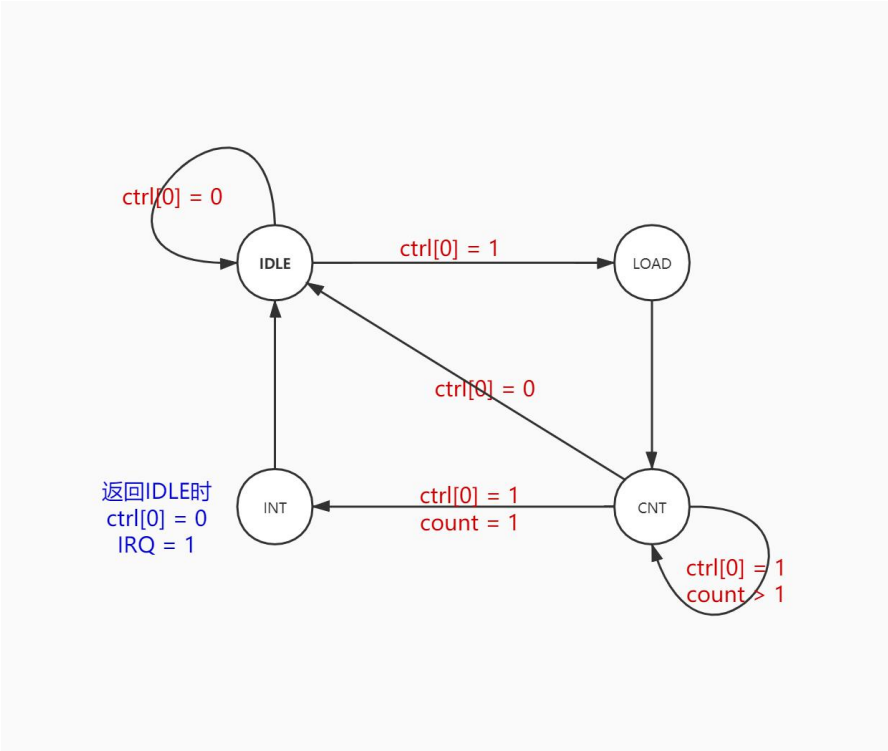
（四）请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态转移图。

答：

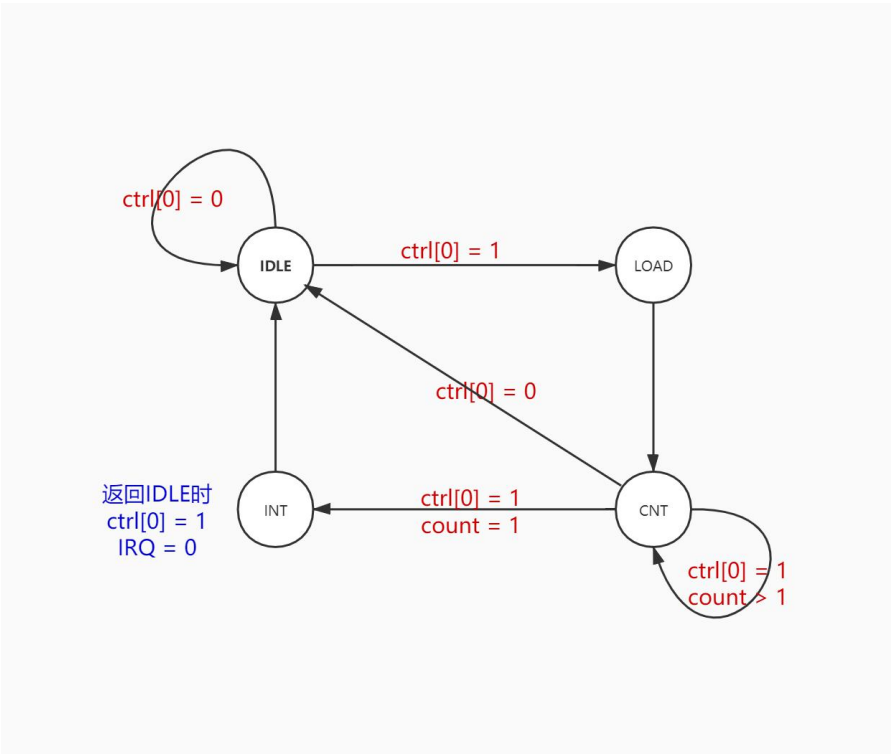
两种中断模式大部分行为是相同的，最初都是处于闲置（IDLE）状态，发出的中断请求保持 0，在 CONTROL 寄存器 0 位置 1 后开始工作，即进入加载（LOAD）状态，将预先设定好的计数初始值（preset）存入 COUNT 寄存器，之后进入计数（CNT）状态，当 Timer 处于计数状态，且 CONTROL 的 0 位（即允许计数）时，count 每周期减一，当 count 减为 0 时，进入中断（INT）状态，产生中断请求（该中断请求信号与 CONTROL 的[3]位与运算（考虑中断屏蔽）得到最终由 Timer 发出的中断信号），在中断状态的下一个周期，状态再次回到闲置状态。

两种中断模式的不同之处就在于进入中断状态后的行为，在 INT 状态中，如果 CONTROL[2:1]=00，即模式 0，但 CONTROL[0]变为 0，即禁止继续计数，状态返回闲置状态，中断请求保持 1，若要使 Timer 重新开始工作，需要再次通过 sw 改变 CONTROL 的值；如果 CONTROL[2:1]=01，即模式 1，中断请求立刻恢复 0，状态返回闲置状态，但此时 CONTROL[0]为 1，Timer 将重新开始计数。

模式 0 状态转移图：



模式 1 状态转移图：



（五）请开发一个主程序以及定时器的 exception handler。

答：

主程序

```
#main
.text
ori $1, $0, 9
ori $2, $0, 0x7f00                  #address of Timer0
ori $3, $0, 0x100                   #for preset
ori $4, $0, 0xfc02                   #for initial SR in CP0
mtc0 $4, $13                       #save SR
sw $3, 4($2)                        #save preset
sw $1, 0($2)                        #save control

Loop:
beq $0, $0, end
nop
end:
```

```
j Loop
nop
```

exception handler

```
#exception handler
.ktext 4180
sw $24, 0($sp)                    #protect now
subi $sp, $sp, 4
sw $25, 0($sp)
subi $sp, $sp, 4
sw $26, 0($sp)
subi $sp, $sp, 4
sw $27, 0($sp)
subi $sp, $sp, 4
sw $28, 0($sp)
subi $sp, $sp, 4
sw $29, 0($sp)
subi $sp, $sp, 4
sw $30, 0($sp)
subi $sp, $sp, 4

mfc0 $26, $13
andi $27, $26, 0x0400                    #interrupt at normal
instruction
andi $28, $26, 0x80000400                 #interrupt at delay
instruction
ori $29, $0, 0x400
ori $30, $0, 0x80000400
beq $27, $29, handler
nop
```

```
beq $28, $30, handler
nop

handler:
ori $25, $0, 9
ori $24, $0, 0x7f00
sw $25, 0($24)                          #save control again

addi $sp, $sp, 4                            #return before
interrupt
lw $30, 0($sp)
addi $sp, $sp, 4
lw $29, 0($sp)
addi $sp, $sp, 4
lw $28, 0($sp)
addi $sp, $sp, 4
lw $27, 0($sp)
addi $sp, $sp, 4
lw $26, 0($sp)
addi $sp, $sp, 4
lw $25, 0($sp)
addi $sp, $sp, 4
lw $24, 0($sp)

eret
```

（六）请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？

答：

点击鼠标或敲击键盘与实验中的 interrupt 信号相似，当鼠标或键盘操作时，

产生外部中断信号，此时程序进入内核态，由于鼠标、键盘的操作行为不同，会产生不同的中断信号，CPU 通过判断这些信号跳转至不同的内核程序，调用不同的内核代码。