计算机组成原理实验报告参考模板

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-COU,支持的指令集包含{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，MTC0,MFC0,ERET}。CPU主要包含了PC,NPC,GRF,ALU,EXT,CTRL,.FWandSCTRL,MUX,MulAndDiv,CP0等模块，同时存储器外置于tb，这些模块按照数据通路、控制信号,冒险控制的设计逐级展开。

（二）关键模块定义

1. GRF

信号名	方向	描述
clk	I	CPU时钟
reset	I	同步复位信号
WE	I	写使能
A1[4:0]	I	rs输入
A2[4:0]	I	rt输入
A3[4:0]	I	rd输入
WD[31:0]	I	寄存器文件写入数据
PC[31:0]	I	current pc
RD1[31:0]	O	读寄存器文件时寄存器rs的输出
RD2[31:0]	O	读寄存器文件时寄存器rt的输出

2. NPC

信号名	方向	描述
PC[31:0]	I	当前PC值
IMM26[25:0]	I	j型指令的26位立即数
IMM16[15:0]	I	b型指令的16位立即数
IMM32[31:0]	I	jr型指令的寄存器数据
NPCOP[2:0]	I	npc控制指令
Zero[4:0]	I	第0位beq信号，1为跳转，0为pc+4；第一位bgez信号，1为跳转，0为pc+4
NPC[31:0]	O	下一个PC的值
PCA4[31:0]	O	当前PC+4

功能定义

OP值	功能
000	pc+4
001	B类指令
010	JAL类指令，跳26Imm
011	JR类指令，跳寄存器

3. ALU

信号名	方向	描述
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
SHAMT[4:0]	I	移位信号
ALUOP[3:0]	I	ALU功能控制
ALUOUT[31:0]	O	ALU计算结果

功能定义

OP值	功能
0000	A+B
0001	A-B
0010	A|B
0011	A&B
0100(lui)	B<<16
0101(SLL)	B<< shamt
0110(SLTI)	若A小于B，则输出1，否则0(有符号)
0111(NOR)	~(A|B)
1000	B<<A[4:0]
1001	若A小于B，则输出1，否则0(无符号)
1010	B>>>A[4:0]
1011	B>>A[4:0]
1100	A^B
1101	B>>>shamt
1110	B>>shamt

CMP

信号名	方向	描述
CMPOP[3:0]	I	控制信号
Rs[31:0]	I	转发后的Rs
Rt[31:0]	I	转发后的Rt
Zero	O	1为真0为假，对于当前B指令

```
BEQ 4'b0000
BGEZ 4'b0001
BLTZ 4'b0010
BGTZ 4'b0011
BLEZ 4'b0100
BNE 4'b0101
```

4.CTRL

信号名	方向	描述
InstrD[31:0]	I	D级指令
NPCOPD[2:0]	O	npc功能控制
ExtopInD[1:0]	O	D级扩展控制
DmweToM	O	数据文件写使能,流水到M级
RFWE	O	grf写使能流水到W级
A3[4:0]	O	根据指令选择的A3
RFWDMUX[2:0]	O	grf写入控制
ALUBMUX[2:0]	O	ALUB端控制
ALUOP[3:0]	O	ALUOP,流水到E
DMOP[2:0]	O	流水到M级控制DM写入
TuseRs[2:0]	O	Rs的Tuse
TuseRt[2:0]	O	Rt的Tuse
TnewE[2:0]	O	指令在E级的Tnew
CMPOP[3:0]	O	CMP控制信号

信号控制矩阵

指令	指令类型	指令码	NPCOP	RFWE	ALUOP	EXTOP	DMWE	A3	ALUBMUX	DMOP	CMPOP	EXTDMOP
ADDU	R	100001	PC+4(000)	1	ADD(0000)	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
SUBU	R	100011	PC+4	1	SUB(0001)	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
ORI	I	001101	PC+4	1	ori(0010)	01	0	InstrD[20:16]	001	XXX	XXXX	XXX
LW	I	100011	PC+4	1	ADD	00	0	InstrD[20:16]	001	XXX	XXXX	XXX
SW	I	101011	PC+4	0	ADD	00	1	XXX	001	000	XXXX	XXX
BEQ	I	000100	B(001)	0	XXXX	XX	0	XXX	000	XXX	0000	XXX
LUI	I	001111	PC+4	1	LUI	01	0	InstrD[20:16]	001	XXX	XXXX}	XXX
SLL	R	000000	PC+4	1	SLL(0101)	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
J	J	000010	010(J)	0	XXXX	XX	0	XXX	XXX	XXX	XXXX	XXX
JAL	J	000011	010	1	XXXX	XX	0	31	XXX	XXX	XXXX	XXX
JR	R	001000	011(JR)	0	XXXX	XX	0	XXX	XXX	XXX	XXXX	XXX
ADDIU	I	001001	PC+4	1	ADD	00	0	InstrD[20:16]	001	XXX	XXXX	XXX
BGEZ	I	000001//00001	001	0	XXXX	00	0	XXX	XXX	XXX	0001	XXX
JALR	R	001001	011	1	XXXX	XX	0	InstrD[15:11]	XXX	XXX	XXXX	XXX
SLTI	I	001010	PC+4	1	0110	00	0	InstrD[20:16]	001	XXX	XXXX	XXX
LB	I	100000	PC+4	1	ADD	00	0	InstrD[20:16]	001	XXX	XXXX	010
SB	I	101000	PC+4	0	ADD	00	1	XXX	001	001	XXXX	XXX
ADD(无溢出)	R	100000	PC+4	1	ADD	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
ADDI(无溢出)	I	001000	PC+4	1	ADD	00	0	InstrD[20:16]	001	XXX	XXXX	XXX

指令	指令类型	指令码	NPCOP	RFWE	ALUOP	EXTOP	DMWE	A3	ALUBMUX	DMOP	CMPOP	EXTDMOP
BLTZ	I	000001//00000	001	0	XXXX	XX	0	XXX	XXX	XXX	0010	XXX
BGTZ	I	000111	001	0	XXXX	XX	0	XXX	XXX	XXX	0011	XXX
BLEZ	I	000110	001	0	XXXX	XX	0	XXX	XXX	XXX	0100	XXX
BNE	I	000101	001	0	XXXX	XX	0	XXX	XXX	XXX	0101	XXX
AND	R	100100	PC+4	1	0011	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
NOR	R	100111	PC+4	1	0111	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
OR	R	100101	PC+4	1	0010	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
SLT	R	101010	PC+4	1	0110	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
SLLV	R	000100	PC+4	1	1000	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
SLTU	R	101011	PC+4	1	1001	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
SRAV	R	000111	PC+4	1	1010	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
SRLV	R	000110	PC+4	1	1011	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
SUB(无溢出)	R	100010	PC+4	1	0001	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
XOR	R	100110	PC+4	1	1100	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
SLTIU	I	001011	PC+4	1	1001	00	0	InstrD[20:16]	001	XXX	XXXX	XXX
SRA	R	000011	PC+4	1	1101	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX

指令	指令类型	指令码	NPCOP	RFWE	ALUOP	EXTOP	DMWE	A3	ALUBMUX	DMOP	CMPOP	EXTDMOP
ANDI	I	001100	PC+4	1	0011	01	0	InstrD[20:16]	001	XXX	XXXX	XXX
SRL	R	000010	PC+4	1	1110	XX	0	InstrD[15:11]	000	XXX	XXXX	XXX
XORI	I	001110	PC+4	1	1100	01	0	InstrD[20:16]	001	XXX	XXXX	XXX
LBU	I	100100	PC+4	1	ADD	00	0	InstrD[20:16]	001	XXX	XXXX	001
LH	I	100001	PC+4	1	ADD	00	0	InstrD[20:16]	001	XXX	XXXX	100
LHU	I	100101	PC+4	1	ADD	00	0	InstrD[20:16]	001	XXX	XXXX	011
SH	I	101001	PC+4	0	ADD	00	1	XXXX	001	010	XXXX	XXX
MFHI	R	010000	PC+4	1	XXXX	XX	0	InstrD[15:11]	XXX	XXX	XXXX	XXX
MFLO	R	010010	PC+4	1	XXXX	XX	0	InstrD[15:11]	XXX	XXX	XXXX	XXX
MULT	R	011000	PC+4	0	XXXX	XX	0	XXXX	XXX	XXX	XXXX	XXX
MULTU	R	011001	PC+4	0	XXXX	XX	0	XXXX	XXX	XXX	XXXX	XXX
MTHI	R	010001	PC+4	0	XXXX	XX	0	XXXX	XXX	XXX	XXXX	XXX
MTLO	R	010011	PC+4	0	XXXX	XX	0	XXXX	XXX	XXX	XXXX	XXX
DIV	R	011010	PC+4	0	XXXX	XX	0	XXXX	XXX	XXX	XXXX	XXX
DIVU	R	011011	PC+4	0	XXXX	XX	0	XXXX	XXX	XXX	XXXX	XXX
MFC0	R	010000/00000	PC+4	1	XXXX	XX	0	InstrD[20:16]	XXX	XXX	XXXX	XXX
MTC0	R	010000/00100	PC+4	0	XXXX	XX	0	XXXX	XXX	XXX	XXXX	XXX
ERET	R	010000/011000	PC+4	0	XXXX	XX	0	XXXX	XXX	XXX	XXXX	XXX

指令	指令类型	指令码	SELOP	RFWDOP	MADOP	InstrMAD	InsOfJump	OVINS	CPZEROWE	EXLCLR
ADDU	R	100001	000	000	0000	0	0	0	0	0
SUBU	R	100011	000	000	0000	0	0	0	0	0
ORI	I	001101	000	000	0000	0	0	0	0	0
LW	I	100011	XXX	001	0000	0	0	0	0	0
SW	I	101011	XXX	XXX	0000	0	0	0	0	0
BEQ	I	000100	XXX	XXX	0000	0	1	0	0	0
LUI	I	001111	000	000	0000	0	0	0	0	0
SLL	R	000000	000	000	0000	0	0	0	0	0
J	J	000010	XXX	XXX	0000	0	1	0	0	0
JAL	J	000011	001	000	0000	0	1	0	0	0
JR	R	001000	XXX	XXX	0000	0	1	0	0	0
ADDIU	I	001001	000	000	0000	0	0	0	0	0
BGEZ	I	000001//00001	XXX	XXX	0000	0	1	0	0	0
JALR	R	001001	001	000	0000	0	1	0	0	0
SLTI	I	001010	000	000	0000	0	0	0	0	0
LB	I	100000	XXX	001	0000	0	0	0	0	0
SB	I	101000	XXX	XXX	0000	0	0	0	0	0
ADD(无溢出)	R	100000	000	000	0000	0	0	1	0	0
ADDI(无溢出)	I	001000	000	000	0000	0	0	1	0	0
BLTZ	I	000001//00000	XXX	XXX	0000	0	1	0	0	0
BGTZ	I	000111	XXX	XXX	0000	0	1	0	0	0
BLEZ	I	000110	XXX	XXX	0000	0	1	0	0	0
BNE	I	000101	XXX	XXX	0000	0	1	0	0	0
AND	R	100100	000	000	0000	0	0	0	0	0
NOR	R	100111	000	000	0000	0	0	0	0	0
OR	R	100101	000	000	0000	0	0	0	0	0
SLT	R	101010	000	000	0000	0	0	0	0	0
SLLV	R	000100	000	000	0000	0	0	0	0	0
SLTU	R	101011	000	000	0000	0	0	0	0	0
SRAV	R	000111	000	000	0000	0	0	0	0	0
SRLV	R	000110	000	000	0000	0	0	0	0	0
SUB(无溢出)	R	100010	000	000	0000	0	0	1	0	0
XOR	R	100110	000	000	0000	0	0	0	0	0
SLTIU	I	001011	000	000	0000	0	0	0	0	0
SRA	R	000011	000	000	0000	0	0	0	0	0
ANDI	I	001100	000	000	0000	0	0	0	0	0
ORI	I	001101	000	000	0000	0	0	0	0	0
SRL	R	000010	000	000	0000	0	0	0	0	0
XORI	I	001110	000	000	0000	0	0	0	0	0
LBU	I	100100	XXX	001	0000	0	0	0	0	0
LH	I	100001	XXX	001	0000	0	0	0	0	0
LHU	I	100101	XXX	001	0000	0	0	0	0	0
SH	I	101001	XXX	XXX	0000	0	0	0	0	0
MFHI	R	010000	010	000	0000	1	0	0	0	0
MFLO	R	010010	011	000	0000	1	0	0	0	0
MULT	R	011000	XXX	XXX	0001	1	0	0	0	0
MULTU	R	011001	XXX	XXX	0010	1	0	0	0	0
MTHI	R	010001	XXX	XXX	0011	1	0	0	0	0

指令	指令类型	指令码	SELOP	RFWDOP	MADOP	InstrMAD	InsOfJump	OVINS	CPZEROWE	EXLCLR
MTLO	R	010011	XXX	XXX	0100	1	0	0	0	0
DIV	R	011010	XXX	XXX	0101	1	0	0	0	0
DIVU	R	011011	XXX	XXX	0110	1	0	0	0	0
MFC0	R	010000/00000	100	000	0000	0	0	0	0	0
MTC0	R	010000/00100	XXX	XXX	0000	0	0	0	1	0
ERET	R	010000/011000	XXX	XXX	0000	0	0	0	0	1

5.FWandSCTRL

信号名	方向	描述
A1D[4:0]	I	
A2D[4:0]	I	
A1E[4:0]	I	
A2E[4:0]	I	
A1M[4:0]	I	
A2M[4:0]	I	
A3M[4:0]	I	
A3W[4:0]	I	
WEE	I	
WEM	I	
WEW	I	
InsrtMADInD	I	D阶段指令是否为乘除指令
BusyOrStart	I	E阶段是否将或正在执行乘除法
TuseRs[2:0]	I	
TuseRt[2:0]	I	
TnewE[2:0]	I	
TnewM[2:0]	I	
FWCMPRS[2:0]	O	cmpRS转发
FWCMPRT[2:0]	O	cmpRT转发
FWALURS[2:0]	O	ALURS转发
FWALURT[2:0]	O	ALURT转发
FWDMRT[2:0]	O	DM写入数据转发
Stall	O	停止信号

T译码	指令
TuseRs	
2	
1	ADDU SUBU ORI LW SW LUI SLTI ADDIU ADD ADDI AND NOR OR SLT SLLV SLTU SRAV SRLV SUB XOR SLTIU ADNI ORI XORI LBU LB LH LHU SH SB MULT MULTU MTHI DIV DIVU
0	BEQ JR JALR BGEZ BLTZ BGTZ BLEZ BNE
TuseRt	
2	SW SH SB
1	ADDU SUBU SLL ADD AND NOR OR SLT SLLV SLTU SRAV SRLV SUB XOR SRA SRL MULT MULTU DIV DIVU MTC0
0	BEQ BNE
TnewE	
2	LW LBU LB LH LHU
1	ADDU SUBU ORI LUI SLL SLTI ADDIU ADD ADDI AND NOR OR SLT SLLV SLTU SRAV SRLV SUB XOR SLTIU SRA ANDI ORI SRL XORI MFHI MFLO MFC0
0	JAL JALR

5.CP0

信号名	方向	描述
clk	I	时钟
Reset	I	复位信号
ReadAddr[4:0]	I	读CP0地址
WriteAddr[4:0]	I	写CP0地址
DataToCP0[31:0]	I	写CP0的数据
IREXPC[31:0]	I	宏观PC
EXCCODE[4:0]	I	异常码
HWINT[5:0]	I	外部中断信号
WE	I	CP0写使能
EXLCLR	I	EXL清0信号
BD	I	延迟槽指令信号
INTREQ	O	CP0发出的异常/中断请求
PCFROMEPC[31:0]	O	EPC的值
DataFromCP0[31:0]	O	读CP0寄存器的值
OUTSIDEINR	O	外部中断信号（HWINT[2]），流水到M级写0X7F20

6.ERRORDECODE

信号名	方向	描述
Instr[31:0]	I	E级指令
ADELOFPC	I	F级流水的取址异常信号
RI	I	D级流水的未知指令信号
OVINS	I	ADD SUB ADDI可能溢出指令信号
OVFROMALU	I	ALU加法溢出信号
ALUOUT[31:0]	I	ALU计算结果
ERRORCODE[4:0]	O	异常译码

```
    assign ERRORCODE = (ADELOFPC)?5'h4:
                       (RI)?5'ha:
                       (OVINS && OVFROMALU)?5'hc:
                       (ADELOFLOAD)?5'h4:
                       (ADES)?5'h5:0;
```

根据异常的优先级译码，F级最优，D级次之，E级最后

7.MulAndDiv

信号名	方向	描述
clk	I	时钟
reset	I	复位信号
MADOP[3:0]	I	控制信号
RS[31:0]	I	E级转发后的RS
RT[31:0]	I	E级转发后的RT
HI[31:0]	O	HI寄存器
LO[31:0]	O	LO寄存器
START	O	有MD指令在E级，准备执行
BUSY	O	MD指令在E级执行中

控制信号	功能
0001	(HI, LO) = GPR[rs]×GPR[rt] 有符号
0010	(HI, LO) = GPR[rs]×GPR[rt] 无符号
0011	HI<=RS
0100	LO<=RS
0101	有符号除
0110	无符号除

8.MUX

DataToGrfMux

```
module DataToGrfMux ( input [31:0] S0,
                      input [31:0] S1,
                      input [31:0] S2,
                      input [31:0] S3,
                      input [31:0] S4,
                      input [31:0] S5,
                      input [31:0] S6,
                      input [31:0] S7,
                      input [2:0] SELOP,
                      output[31:0] DATATOGRF

);
    DataToGrfMux CPUDataToGrfMuxINE(.S0(ALUOUTINE),
                                    .S1(PCE+8),
                                    .S2(HIFROMMAD),
                                    .S3(LOFROMMAD),
```

```
                                    .S4(0),
                                    .S5(0),
                                    .S6(0),
                                    .S7(0),
                                    .SELOP(SELOPINE),
                                    .DATATOGRF(DATATOGRFINE)

    );
```

```
SELOP
000     ALU输出
001     PC+8
010     HI
011     LO
100     CP0
```

RFWDMUX

信号名	方向	描述
RFWDOP[2:0]	I	控制信号
ALUOUT[31:0]	I	ALU计算结果
DMOUT[31:0]	I	数据存储器的输出数据
PCA4[31:0]	O	PC+4
RFWD[31:0]	O	写入GRF的数据

选择	功能
000	M级写入寄存器的数据
001	DMOUT

ALUBMUX

信号名	方向	描述
ALUBOP[2:0]	I	控制信号
rt[31:0]	I	转发后RT寄存器的值
IMM16[31:0]	I	经过EXT扩展后的16位立即数
ALUB[31:0]	O	ALUB的输入

选择	功能
000	rt
001	imm16(I型指令)

DMMUX

信号名	方向	描述
DMWE	I	是否要写
Addr[1:0]	I	地址低两位
DMMUXOP	I	sw sb sh的选择
DMOP[3:0]	O	输出到DM的写使能

选择	功能
000	sw
001	sb
010	sh

EXTDMMUX

信号名	方向	描述
Addr[1:0]	I	地址低两位
DataFromDM[31:0]	I	从DM读出的数据
EXTDMOP[2:0]	I	数据扩展选择信号
Dout[31:0]	O	扩展后的数据

```
000：无扩展
001：无符号字节数据扩展
010：符号字节数据扩展
011：无符号半字数据扩展
100：符号半字数据扩展
```

9. DM

信号名	方向	描述
DMOP[2:0]	I	写入数据方式选择
Address[31:0]	I	数据存储地址
Input[31:0]	I	输入数据
clk	I	全局时钟
Reset	I	同步清零
DMWE	I	写使能
PC[31:0]	I	pc
Data[31:0]	O	输出数据

选择	功能
000	32imm
001	sb8位

10. EXT

信号名	方向	描述
EXTOP[2:0]	I	选择信号
imm16[15:0]	I	16位立即数
shamt[4:0]	I	5位立即数
extout[31:0]	O	输出

选择	功能
00	符号扩展16-32
01	无符号扩展16-32

11. IFU(分为PC和IM)

- PC

信号名	方向	描述
npc[31:0]	I	下个PC值
reset	I	同步复位至0x0000_3000
clk	I	时钟
pc[31:0]	O	当前PC

- IM

信号名	方向	描述
Addr[23:0]	I	当前指令地址
instr[31:0]	O	当前指令

12.流水线寄存器

```
module PipeD (
    input  clk,
    input reset,
    input stall,
    input [31:0] InstrInF,
    input [31:0] PCA4F,
    input [31:0] PCF,
    input ADELOFPCINF,
    input BDINF,
    output reg [31:0] PCD,
    output reg [31:0] PCA4D,
    output reg [31:0] InstrInD,
    output reg  ADELOFPCIND,
    output reg  BDIND
);
```

```
module PipeE (
    input clk,
    input reset,
    input [31:0] PCD,
    input [31:0] InstrD,
    input RFWEInD,
    input DMWEInD,
    input [4:0] A3InD,
    input [31:0] IMM16EXTD,
    input [2:0] TnewEinD,
    input [2:0] RFWDMUXIND,
    input [2:0] ALUBMUXIND,
    input [3:0] ALUOPIND,
    input [1:0] EXTOPIND,
    input [2:0] DMOPIND,
    input [31:0] RsInD,
    input [31:0] RtInD,
    input [2:0] EXTDMOPIND,
    input [2:0] SELOPIND,
    input [3:0] MADOPD,
    input condWinD,
    input ADELOFPCIND,
    input BDIND,
    input OVINSIND,
    input UNKNOWNINSIND,
    input CPZEROWEIND,
    input EXLCLRIND,
    output reg [31:0] PCE,
    output reg [31:0] InstrE,
    output reg RFWEInE,
    output reg DMWEInE,
    output reg [4:0] A3InE,
```

```
    output reg [31:0] IMM16EXTE,
    output reg [2:0] TnewE,
    output reg [2:0] RFWDMUXINE,
    output reg [2:0] ALUBMUXINE,
    output reg [3:0] ALUOPINE,
    output reg [1:0] EXTOPINE,
    output reg [2:0] DMOPINE,
    output reg [31:0] RsInE,
    output reg [31:0] RtInE,
    output reg [2:0] EXTDMOPINE,
    output reg [2:0] SELOPINE,
    output reg [3:0] MADOPE,
    output reg condWinE,
    output reg ADELOFPCINE,
    output reg BDINE,
    output reg OVINSINE,
    output reg UNKNOWNINSINE,
    output reg CPZEROWEINE,
    output reg EXLCLRINE
);
```

```
module PipeM (
    input clk,
    input reset,
    input [31:0] PCE,
    input [31:0] InstrE,
    input RFWEInE,
    input DMWEInE,
    input [4:0] A3InE,
    input [2:0] TnewE,
    input [2:0] RFWDMUXINE,
    input [2:0] DMOPINE,
    input [31:0] RtFromE,
    input [31:0] ALUOUTINE,
    input [2:0] EXTDMOPINE,
    input [31:0] DATATOGRFINE,
    input condWinE,
    input OUTSIDEINRINE,
    output reg [31:0] PCM,
    output reg [31:0] InstrM,
    output reg RFWEInM,
    output reg DMWEInM,
    output reg [4:0] A3InM,
    output reg [2:0] TnewM,
    output reg [2:0] RFWDMUXINM,
    output reg [2:0] DMOPINM,
    output reg [31:0] RtInM,
    output reg [31:0] ALUOUTINM,
    output reg [2:0] EXTDMOPINM,
    output reg [31:0] DATATOGRFINM,
    output reg condWinM,
    output reg OUTSIDEINRINM
);
```

```
module PipeW (
    input clk,
    input reset,
    input [31:0] PCM,
    input [31:0] InstrM,
    input RFWEInM,
    input [4:0] A3InM,
    input [2:0] RFWDMUXINM,
    input [31:0] ALUOUTINM,
    input [31:0] DMOUTM,
    input [31:0] DATATOGRFINM,
    output reg [31:0] PCW,
    output reg [31:0] InstrW,
    output reg RFWEInW,
    output reg [4:0] A3InW,
    output reg [2:0] RFWDMUXINW,
    output reg [31:0] ALUOUTINW,
    output reg [31:0] DMOUTW,
    output reg [31:0] DATATOGRFINW
);
```

10.转发Mux

```
module  FWCMPRS(input [2:0] FWOP,
                input [31:0] DataFromE,
                input [31:0] DataFromM,
                input [31:0] DataFromW,
                input [31:0] DataFromD,
                output [31:0] RsToCmp
);
```

```
module  FWCMPRT(input [2:0] FWOP,
                input [31:0] DataFromE,
                input [31:0] DataFromM,
                input [31:0] DataFromW,
                input [31:0] DataFromD,
                output [31:0] RtToCmp
);
```

```
module FWALURS (input [2:0] FWOP,
                input [31:0] DataFromM,
                input [31:0] DataFromW,
                input [31:0] DataFromE,
                output [31:0] RsToAlu

);
```

```
module FWALURT(input [2:0] FWOP,
               input [31:0] DataFromM,
               input [31:0] DataFromW,
               input [31:0] DataFromE,
               output [31:0] RtToAlu

);
```

```
module FWDRT (input [2:0] FWOP,
              input [31:0] DataFromM,
              input [31:0] DataFromW,
              output [31:0] WDToDM

);
```

Bridge

信号名	方向	描述
AddrFromCPU [31:0]	I	CPU传来的地址
WEFromCPU	I	CPU的写使能
DataFromCPU [31:0]	I	CPU传来的数据
DataToCPUFromDM[31:0]	I	DM传来的数据
DataToCPUFromTime0[31:0]	I	Time0传来的数据
DataToCPUFromTime1 [31:0]	I	Time1传来的数据
DMWE	O	桥传出的DM写使能
Time0WE	O	桥传出的Time0写使能
Time1WE	O	桥传出的Time1写使能
AddrToDM [31:0]	O	桥传给DM的地址
AddrToTime0 [31:0]	O	桥传给Time0的地址
AddrToTime1 [31:0]	O	桥传给Time0的地址
DataToDM [31:0]	O	桥传给DM的数据
DataToTime0 [31:0]	O	桥传给Time0的数据
DataToTime1 [31:0]	O	桥传给Time1的数据
DataToCPU[31:0]	O	桥传给CPU的数据

```
    //DM
    assign AddrToDM = AddrFromCPU;
    assign DMWE = WEFromCPU & HITDM;
    assign DataToDM = DataFromCPU;

    //Time0
```

```
    assign AddrToTime0 = AddrFromCPU;
    assign Time0WE = WEFromCPU & HITTIME0;
    assign DataToTime0 = DataFromCPU;

    //Time1
    assign AddrToTime1 = AddrFromCPU;
    assign Time1WE = WEFromCPU & HITTIME1;
    assign DataToTime1 = DataFromCPU;

    //DataToCPU
    assign DataToCPU = HITDM?(DataToCPUFromDM):
                       HITTIME0?(DataToCPUFromTime0):
                       HITTIME1?(DataToCPUFromTime1):0;
```

（三）重要机制实现方法

1. 跳转

NPC模块在D级传递给CMP和NPC控制信号，CMP控制B类指令的Zero信号，CMPOP控制CMP判断的是哪条B指令的Zero,Zero信号传递到NPC中协同B指令跳转。NPC信号选择是什么类型的跳转，类型有J(跳26imm),JR(跳转发后的Rs,无转发视为原地转发)，B(跳imm16，zero判断是否跳),和F级PC+4.

2. 流水线延迟槽

跳转指令在D级时，PC+4的指令在F级，跳转指令只更新PC值，PC+4指令进入D级，该指令即为延迟槽。同时JAL应用PC+8写入$31

3. 转发和暂停

在D级的CTRL模块对D级指令进行AT译码，将TuseRs和TuseRt和A3D送入转发暂停单元.

```
module FWandSCTRL(
    input [4:0] A1D,
    input [4:0] A2D,
    input [4:0] A1E,
    input [4:0] A2E,
    input [4:0] A1M,
    input [4:0] A2M,
    input [4:0] A3E,
    input [4:0] A3M,
    input [4:0] A3W,
    input WEE,
    input WEM,
    input WEW,
    input [2:0] TuseRs,
    input [2:0] TuseRt,
    input [2:0] TnewE,
    input [2:0] TnewM,
    output [2:0] FWCMPRS,
    output [2:0] FWCMPRT,
    output [2:0] FWALURS,
    output [2:0] FWALURT,
    output [2:0] FWDMRT,
    output Stall
    );  // FW
    //FWCMP
```

```
    assign FWCMPRS = (A1D==A3M && WEM && A3M)?`CMPFROMM:
                     (A1D==A3W && WEW && A3W)?`CMPFROMW:`CMPFROMD;

    assign FWCMPRT = (A2D==A3M && WEM && A3M)?`CMPFROMM:
                     (A2D==A3W && WEW && A3W)?`CMPFROMW:`CMPFROMD;

    //FWALURS
    assign FWALURS = (A1E==A3M && WEM && A3M)?`ALUFROMM:
                     (A1E==A3W && WEW && A3W)?`ALUFROMW:`ALUFROME;

    assign FWALURT = (A2E==A3M && WEM && A3M)?`ALUFROMM:
                     (A2E==A3W && WEW && A3W)?`ALUFROMW:`ALUFROME;

    //FWDMRT
    assign FWDMRT = (A2M==A3W && WEW && A3W)?1:0;


    //Stall
    wire StallRsE,StallRsM,StallRtE,StallRtM;
    assign StallRsE = (TuseRs<TnewE) && (A1D) && (A1D==A3E) &&WEE;
    assign StallRsM = (TuseRs<TnewM) && (A1D) && (A1D==A3M) && WEM;
    assign StallRtE = (TuseRt<TnewE) && (A2D) && (A2D==A3E) && WEE;
    assign StallRtM = (TuseRt<TnewM) && (A2D) && (A2D==A3M) && WEM;
    assign Stall = StallRsE|StallRsM|StallRtE|StallRtM;

endmodule
```

根据DEMW级的A1A2A3和WE信号转发，通过Tuse和Tnew控制暂停

4.IO设计

CPU通过系统桥来访问外设(Time0,Time1)，进行数据的存入和读出，同时通过HWINT接受外部的中断信号

二、测试方案

（一）典型测试样例

```
ori $28, 0
ori $29, 0
#data
li $8, 1
li $9, 11
li $10, 111
li $11, 1111
li $12, 11111
li $13, 111111
li $14, 1111111
li $15, 11111111
##addr
ori $16, 0
ori $17, 20
ori $18, 40
ori $19, 60
ori $20, 80
ori $21, 100
sw $8, 0($16)
```

```
sw $9, 4($16)
sw $10, 8($16)
sw $11, 12($16)
sw $12, 16($16)
sw $13, 20($16)
sw $14, 24($16)
sw $15, 28($16)
##zhuanfa
## 1
lw $1, 0($16)
sw $1, 8($17)
nop
## 2
lw $2, 4($16)
addu $10, $10, $11
sw $2, 0($18)
# 3
addu $1, $3, $4
subu $2, $4, $3
ori $3, 151
sw $1, 16($16)
#stall
#4
ori $4, 4
sw $4, 0($0)
nop
nop
nop
##
lw $17, 0($0)
sw $4, 0($17)
##
#5
addu $7, $8, $9
##
addu $6, $8, $9
beq $6, $7, branch1
##
subu $3, $14, $15
nop
ori $25, 56
branch1:
#6
jal branch2
addu $31, $31, $31
#
nop
branch2:
subu $31, $31, $26
ori $31, 0x30e8
nop
nop
nop
nop
##
lw $4, 0($0)
beq $6, $4, branch2
##
```

```
nop
lw $4, 0($0)
ori $1, 0
beq $6, $4, branch2
```

```
li $1, 848
li $2, 8481
li $3, 447
li $4, 4878
li $5, 4876
li $6, 8979
li $7, 9741
li $8, 4871
li $9, 1409
li $10, 454
li $11, 47
li $12, 4721
li $13, 48787
li $14, 5989
li $15, 5987
li $16, 487
li $17, 5978
li $18, 47
li $19, 2021
li $20, 497
li $21, 587
li $22, 48
li $23, 488
li $24, 878
li $25, 487
li $26, 87
li $27, 481
li $28, 484
li $29, 4184
li $30, 48
li $31, 878
sb $1, 0($0)
sb $2, 1($0)
sb $3, 2($0)
sb $4, 3($0)
lh $4, 2($0)
sw $4, 0($0)
sh $5, 2($0)
sh $6, 4($0)
sh $7, 6($0)
lb $7, 5($0)
lb $7, 6($0)
lb $7, 7($0)
lb $8, 4($0)
sh $7, 6($0)
lbu $7, 5($0)
lbu $7, 6($0)
lbu $7, 7($0)
lbu $8, 4($0)
lhu $8, 6($0)
jal ddd
add $31, $1, $2
```

```
addi $15, $15, 1
ddd:
sw $31, 8($0)
jal d1
sw $31, 12($0)
lw $31, 0($0)
d1:
beq $31, $0,d2
li $31, 2
addi $31, $31, 5
d2:
li $31, 0
```

外部中断

```
    wire [31:0] fixed_macroscopic_pc;

    assign fixed_macroscopic_pc = macroscopic_pc & 32'hfffffffc;

    always @(posedge clk) begin
        if (reset) interrupt <= 0;
    end

    always @(negedge clk) begin
        if (~reset && interrupt && |m_data_byteen) begin
            if (fixed_addr == 32'h7F20) begin
                interrupt <= 0;
            end
        end
    end

    reg need_interrupt = 9;

    always @(negedge clk) begin
        if (~reset) begin
            if (need_interrupt && fixed_macroscopic_pc == 32'h3010) begin
                interrupt <= 1;
                need_interrupt <= need_interrupt - 1;
            end
            else if(need_interrupt && fixed_macroscopic_pc == 32'h3038) begin
                interrupt <= 1;
                need_interrupt <= need_interrupt - 1;
            end
            else if (need_interrupt && fixed_macroscopic_pc == 32'h3100) begin
                interrupt <= 1;
                need_interrupt <= need_interrupt - 1;
            end
            else if (need_interrupt && fixed_macroscopic_pc == 32'h3114) begin
                interrupt <= 1;
                need_interrupt <= need_interrupt - 1;
            end
            else if (need_interrupt && fixed_macroscopic_pc == 32'h3220) begin
                interrupt <= 1;
                need_interrupt <= need_interrupt - 1;
            end
            else if (need_interrupt && fixed_macroscopic_pc == 32'h3234) begin
```

```
                interrupt <= 1;
                need_interrupt <= need_interrupt - 1;
            end
            else if (need_interrupt && fixed_macroscopic_pc == 32'h3310) begin
                interrupt <= 1;
                need_interrupt <= need_interrupt - 1;
            end
            else if (need_interrupt && fixed_macroscopic_pc == 32'h4180) begin
                interrupt <= 1;
                need_interrupt <= need_interrupt - 1;
            end
            else if (need_interrupt && fixed_macroscopic_pc == 32'h4188) begin
                interrupt <= 1;
                need_interrupt <= need_interrupt - 1;
            end
        end
    end
```

计数器(模式0)

```
.text
li $t0, 0xfc01
mtc0 $t0, $12
li $t0, 9 ## moshi 0
sw $t0, 0x7f00($0)
li $t0, 114514
sw $t0, 0x7f04($0)
li $1, 0x7fffffff
li $2, 0x7fffffff
add $3, $1, $2
nop
nop
nop
li $1, 1
li $2, 2
li $3, 3
li $4, 4
li $5, 5
li $6, 6
.ktext 0x4180
li $t0, 9
sw $t0, 0x7f00($0)
li $1, 0
li $2, 0
eret
```

计数器(模式1)

```
.text
li $t0, 0xfc01
mtc0 $t0, $12
li $t0, 11 ## moshi 1
sw $t0, 0x7f10($0)
li $t0, 114514
sw $t0, 0x7f04($0)
```

```
li $1, 0x7fffffff
li $2, 0x7fffffff
add $3, $1, $2
nop
nop
nop
li $1, 1
li $2, 2
li $3, 3
li $4, 4
li $5, 5
li $6, 6
.ktext 0x4180
li $t0, 11
sw $t0, 0x7f00($0)
li $1, 0
li $2, 0
eret
```

异常

```
.text
li $t0, 0xfc01
mtc0 $t0, $12
li $t0, 11 ## moshi 1
sw $t0, 0xffffff($0) ##ADES
li $t0, 114514
sw $t0, 0x7f10($0)
li $1, 0x7fffffff
li $2, 0x7fffffff
add $3, $1, $2
nop
nop
nop
li $1, 1
li $2, 2
li $3, 3
li $4, 4
li $5, 5
li $6, 6
.ktext 0x4180
li $1, 0
li $2, 0
eret
```

```
.text
li $t0, 0xfc01
mtc0 $t0, $12
li $t0, 11 ## moshi 1
lw $t0, 0xffffff($0) ADEL
li $t0, 114514
sw $t0, 0x7f00($0)
li $1, 0x7fffffff
li $2, 0x7fffffff
add $3, $1, $2
nop
```

```
nop
nop
li $1, 1
li $2, 2
li $3, 3
li $4, 4
li $5, 5
li $6, 6
.ktext 0x4180
li $1, 0
li $2, 0
eret
```

```
li $1, 1
jr $1
nop
nop
nop
```

（二）自动测试工具

讨论区

三、思考题

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:软件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。

硬件接口指的是两个硬件设备之间的连接方式。硬件接口既包括物理上的接口，还包括逻辑上的数据传送协议。

2. BE 部件对所有的外设都是必要的吗?

答:不一定必要，如果外设只能按字读取，则不必要。或者有些外设不能被CPU写入数据，此时也无需BE。

3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:

模式0:

当计数器倒计数为 0 后，计数器停止计数，此时控制寄存器中的使能 Enable自动变为 0。当使能 Enable 被设置为 1 后，初值寄存器值再次被加载至计数器，计数器重新启动倒计数。 模式 0 通常用于产生定时中断。例如，为操作系统的时间片调度机制提供定时。模式 0 下的中断信号将持续有效，直至控制寄存器中的中断屏蔽位被设置为0。

模式1:

当计数器倒计数为 0 后，初值寄存器值被自动加载至计数器，计数器继续倒计数。模式1通常用于产生周期性脉冲。例如，可以用模式 1产生步进电机所需的步进控制信号。不同于模式 0，模式1下计数器每次计数循环中只产生一周期的中断信号。

状态转移图见计数器文档。

4、请开发一个主程序以及定时器的 exception handler。整个系统完成如下功能：

（1）定时器在主程序中被初始化为模式 0；

（2）定时器倒计数至 0 产生中断；

（3）handler 设置使能 Enable 为 1 从而再次启动定时器的计数器。(2) 及 (3) 被无限重复。

（4）主程序在初始化时将定时器初始化为模式 0，设定初值寄存器的初值为某个值，如 100 或 1000。(注意，主程序可能需要涉及对 CP0.SR 的编程，推荐阅读过后文后再进行。)

```
.text
li $t0, 0xfc01
mtc0 $t0, $12
li $t0, 9
sw $t0, 0x7f00($0)
li $t0, 114514
sw $t0, 0x7f04($0)

.ktext 0x4180
li $t0, 9
sw $t0, 0x7f00($0)
eret
```

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:键盘敲一下和鼠标点一下会向CPU发送外部中断信号，CPU根据不同的中断信号进入不同的中断执行程序，将对应的按键数据读取进入寄存器，然后进行相关操作。