计算机组成原理实验报告

一、CPU 设计方案综述

总体设计综述

本 **CPU** 为 **Verilog** 实现的流水线 **CPU**，支持指令集包含

```
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
```

为了实现这些功能，**CPU** 主要包含了

```
ALU,Bridge,Comp,CP0,CPU,Ctrl,Datapath,DM,Ext,Forward,GRF,Hazard,IFU,IM,LoadData,MDU,PC,Timer
```

这些模块并分成 5 级，并支持 5 个异常和中断处理。

Exccode	助记符	描述
0	Int	中断
4	AdEL	取数或取指异常
5	AdES	存数异常
10	RI	未知指令
12	Ov	溢出异常

关键模块定义

IFU

信号名	方向	信号描述
`clk`	I	时钟信号
`reset`	I	同步复位信号
`stall`	I	暂停信号
`Branch`	I	分支信号
`Jump`	I	跳转信号
`CP0Req`	I	发生异常中断信号
`EXLclr`	I	`eret` 检测信号
`PCBranch[31:0]`	I	分支地址
`PCJump[31:0]`	I	跳转地址
`EPC[31:0]`	I	`EPC` 地址
`Instr[31:0]`	O	输出指令
`Pc[31:0]`	O	输出地址

序号	功能名称	功能描述
1	取指令	根据 `PC` 地址将指令读出
2	复位	当复位信号有效时，将 `PC` 设置为 `0x00003000`
3	暂停	将 `PC` 暂停，输出指令不变
4	计算下一个 `PC`	若 `Jump` 有效，则 `PC = JumpAddr` 若 `Branch` 有效，则 `PC = BranchAddr` 否则，`PC = PC + 4`
5	异常中断相应	`CP0req` 有效跳转至 `0x00004180`

IM

信号名	方向	信号描述
`PC[31:0]`	I	当前 `PC` 地址
`Instr[31:0]`	O	对应地址的指令

PC

信号名	方向	信号描述
`clk`	I	时钟信号
`reset`	I	同步复位信号
`stall`	I	暂停信号
`NPC_Sel`	I	控制下一个 `PC` 信号
`NPC[31:0]`	I	`Branch` 或 `Jump` 的 `PC` 信号
`PC[31:0]`	O	计算后的 `PC`

ALU

信号名	方向	信号描述
`sign`	I	符号溢出判断
`AluCtrl[3:0]`	I	控制运算方式信号
`A[31:0]`	I	第一运算数
`B[31:0]`	I	第二运算数
`S[4:0]`	I	移位数
`D[31:0]`	O	运算结果数

序号	功能名称	功能描述
1	算术运算	将 `A` 和 `B` 进行算术运算
2	按位逻辑运算	将 `A` 和 `B` 按位进行逻辑运算
3	移位运算	将 `B` 按照 `S` 的移位数进行移位

MDU

信号名	方向	信号描述
`clk`	I	时钟信号
`reset`	I	复位信号
`start`	I	乘除指令信号
`MDU_OP[3:0]`	I	`MDU` 操作信号
`A[31:0]`	I	第一运算数
`B[31:0]`	I	第二运算数
`Busy`	O	繁忙信号
`Hi[31:0]`	O	`HI` 寄存器值
`LO[31:0]`	O	`LO` 寄存器值

序号	功能名称	功能描述
1	复位	当复位信号有效时，将 `HI` 和 `LO` 清零
2	乘除	若进行乘法运算，高 32 位存入 `HI` 寄存器，低 32 位存入 `LO` 寄存器 若进行除法运算，$A \mod B$ 存入 `HI` 寄存器，$A \div B$ 存入 `LO` 寄存器
3	取数	读取指定寄存器
4	存数	将 A 写入指定寄存器

Ctrl

信号名	方向	信号描述
`Instr[31:0]`	I	当前指令
`RegDst[1:0]`	O	选择 `rd` 、 `rt` 或 `$31` 作为写入目标寄存器
`Ext_Op[1:0]`	O	控制扩展方式
`Branch[2:0]`	O	检测条件偏移指令
`AluCtrl[3:0]`	O	ALU 运算方式
`MDU_OP[3:0]`	O	MDU 运算方式
`AluSrc`	O	控制参与运算的数值
`RegWrite`	O	寄存器堆写入使能
`MemToReg`	O	控制 `DM` 数据写入寄存器堆
`MemWrite`	O	`DM` 写入使能
`Jump`	O	检测 `J` 指令信号
`Link`	O	将 `PC + 4` 写入寄存器信号
`Return`	O	将寄存器的地址写入 `PC` 信号
`start`	O	乘除指令信号
`ifmfhi`	O	`mfhi` 信号
`ifmflo`	O	`mflo` 信号
`ifmdu`	O	乘除槽信号
`Zero`	O	比较 0 信号
`iflw`	O	`lw` 信号
`iflh`	O	`lh` `lhu` 信号
`iflb`	O	`lb` `lbu` 信号
`ifsw`	O	`sw` 信号
`ifsh`	O	`sh` 信号
`ifsb`	O	`sb` 信号
`sign`	O	溢出判断使能信号
`cal`	O	`add` `addi` `sub` 指令信号
`DelaySlot`	O	受害延迟槽信号
`CP0Write`	O	`CP0` 写入使能信号
`CP0Src`	O	控制 `CP0` 数据信号
`EXLclr`	O	`eret` 指令信号
`unknown_Instr`	O	未知指令信号
`Tuse_rs[2:0]`	O	使用 `rs` 的最大级数
`Tuse_rt[2:0]`	O	使用 `rt` 的最大级数
`Tnew[2:0]`	O	最新数据产生的级数
`LoadType[2:0]`	O	控制 `Load` 指令输出数据信号
`MuxData[2:0]`	O	E 级数据控制转发数据信号

Instr	RegDst	Ext_Op	Branch	AluCtrl	MDU_OP	AluSrc	Regwrite	MemToReg	Memwrite	Jump	Link	Return	start	ifmfhi	ifmflo	ifmdu	Zero	Tuse_rs	Tuse_rt	Tnew	LoadType	MuxData
add	rd	x	x	add	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
addu	rd	x	x	add	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
and	rd	x	x	and	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
sub	rd	x	x	sub	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
subu	rd	x	x	sub	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
jr	x	x	x	x	x	0	0	0	0	1	0	1	0	0	0	0	0	D	x	x	x	x
jalr	rd	x	x	x	x	0	1	0	0	1	1	1	0	0	0	0	0	D	x	F	x	x
or	rd	x	x	or	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
xor	rd	x	x	xor	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
nor	rd	x	x	nor	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
sll	rd	x	x	sll	x	0	1	0	0	0	0	0	0	0	0	0	0	x	E	E	x	ALU
srl	rd	x	x	srl	x	0	1	0	0	0	0	0	0	0	0	0	0	x	E	E	x	ALU
sra	rd	x	x	sra	x	0	1	0	0	0	0	0	0	0	0	0	0	x	E	E	x	ALU
slt	rd	x	x	slt	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
sltu	rd	x	x	sltu	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
sllv	rd	x	x	sllv	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
srlv	rd	x	x	srlv	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
srav	rd	x	x	srav	x	0	1	0	0	0	0	0	0	0	0	0	0	E	E	E	x	ALU
mult	x	x	x	x	mult	0	0	0	0	0	0	0	1	0	0	1	0	E	E	x	x	x
multu	x	x	x	x	multu	0	0	0	0	0	0	0	1	0	0	1	0	E	E	x	x	x
div	x	x	x	x	div	0	0	0	0	0	0	0	1	0	0	1	0	E	E	x	x	x
divu	x	x	x	x	divu	0	0	0	0	0	0	0	1	0	0	1	0	E	E	x	x	x
mthi	x	x	x	x	mthi	0	0	0	0	0	0	0	0	0	0	1	0	E	x	x	x	x
mtlo	x	x	x	x	mtlo	0	0	0	0	0	0	0	0	0	0	1	0	E	x	x	x	x
mfhi	rd	x	x	x	x	0	1	0	0	0	0	0	0	1	0	1	0	x	x	E	x	HI
mflo	rd	x	x	x	x	0	1	0	0	0	0	0	0	0	1	1	0	x	x	E	x	LO
addi	rt	sign	x	add	x	1	1	0	0	0	0	0	0	0	0	0	0	E	x	E	x	ALU
addiu	rt	sign	x	add	x	1	1	0	0	0	0	0	0	0	0	0	0	E	x	E	x	ALU
andi	rt	zero	x	and	x	1	1	0	0	0	0	0	0	0	0	0	0	E	x	E	x	ALU
ori	rt	zero	x	or	x	1	1	0	0	0	0	0	0	0	0	0	0	E	x	E	x	ALU
xori	rt	zero	x	xor	x	1	1	0	0	0	0	0	0	0	0	0	0	E	x	E	x	ALU
lw	rt	sign	x	add	x	1	1	1	0	0	0	0	0	0	0	0	0	E	x	M	word	x
lb	rt	sign	x	add	x	1	1	1	0	0	0	0	0	0	0	0	0	E	x	M	sign_byte	x
lbu	rt	sign	x	add	x	1	1	1	0	0	0	0	0	0	0	0	0	E	x	M	byte	x
lh	rt	sign	x	add	x	1	1	1	0	0	0	0	0	0	0	0	0	E	x	M	sign_half	x
lhu	rt	sign	x	add	x	1	1	1	0	0	0	0	0	0	0	0	0	E	x	M	half	x
lui	rt	upper	x	add	x	1	1	0	0	0	0	0	0	0	0	0	0	x	x	D	x	x
sw	x	sign	x	add	x	1	0	0	1	0	0	0	0	0	0	0	0	E	M	x	x	x
sb	x	sign	x	add	x	1	0	0	1	0	0	0	0	0	0	0	0	E	M	x	x	x
sh	x	sign	x	add	x	1	0	0	1	0	0	0	0	0	0	0	0	E	M	x	x	x
j	x	x	x	x	x	0	0	0	0	1	0	0	0	0	0	0	0	x	x	x	x	x
jal	$31	x	x	x	x	0	1	0	0	1	1	0	0	0	0	0	0	x	x	F	x	x
beq	x	sign	010	x	x	0	0	0	0	0	0	0	0	0	0	0	0	D	D	x	x	x
bne	x	sign	101	x	x	0	0	0	0	0	0	0	0	0	0	0	0	D	D	x	x	x
slti	rt	sign	x	slt	x	1	1	0	0	0	0	0	0	0	0	0	0	E	x	E	x	ALU
sltiu	rt	sign	x	sltu	x	1	1	0	0	0	0	0	0	0	0	0	0	E	x	E	x	ALU
bgez	x	sign	110	x	x	0	0	0	0	0	0	0	0	0	0	0	1	D	x	x	x	x
bgtz	x	sign	100	x	x	0	0	0	0	0	0	0	0	0	0	0	1	D	x	x	x	x
blez	x	sign	011	x	x	0	0	0	0	0	0	0	0	0	0	0	1	D	x	x	x	x
bltz	x	sign	001	x	x	0	0	0	0	0	0	0	0	0	0	0	1	D	x	x	x	x
eret	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x
mfc0	rt	x	x	x	x	0	0	1	0	0	0	0	0	0	0	0	0	x	x	M	x	x
mtc0	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	x	M	x	x	x

Datapath

信号名	方向	信号描述
clk	I	时钟信号
reset	I	同步复位信号
stall_PC	I	PC 暂停信号
stall_FD	I	FD 寄存器暂停信号
stall_DE	I	DE 寄存器暂停信号
stall_EM	I	EM 寄存器暂停信号
stall_MW	I	MW 寄存器暂停信号
clr_FD	I	FD 寄存器清空信号
clr_DE	I	DE 寄存器清空信号
clr_EM	I	EM 寄存器清空信号
clr_MW	I	MW 寄存器清空信号
ForwardSel_D1[4:0]	I	D 级 RD1 转发信号
ForwardSel_D2[4:0]	I	D 级 RD2 转发信号
ForwardSel_EA[4:0]	I	E 级 A 转发信号
ForwardSel_EB[4:0]	I	E 级 B 转发信号
ForwardSel_MD[4:0]	I	M 级写入数据转发信号
PrRD[31:0]	I	Timer 读入数据
HWInt[15:10]	I	中断信号
Branch_D[2:0]	I	D 级 Branch 判断信号
ExtCtrl_D[1:0]	I	D 级扩展控制信号
RegDst[1:0]	I	D 级 A3 控制信号
Jump_D	I	D 级跳转信号
Return_D	I	D 级跳转寄存器信号
RegWrite_D	I	D 级写入使能信号
unknown_Instr_D	I	未知指令信号
AluCtrl_E[3:0]	I	E 级 ALU 控制运算信号
MDU_OP_E[3:0]	I	E 级乘除槽控制运算信号
AluSrc_E	I	E 级控制运算数据信号
ifmfhi_E	I	E 级 mfhi 信号
ifmflo_E	I	E 级 mflo 信号
start_E	I	E 级乘除信号
sign_E	I	溢出检测信号
cal_E	I	add addi sub 指令信号
iflw_E	I	lw 信号
iflh_E	I	lh lhu 信号
iflb_E	I	lb lbu 信号
ifsw_E	I	sw 信号
ifsh_E	I	sh 信号
ifsb_E	I	sb 信号
MemWrite_M	I	M 级 DM 写入使能信号
CP0Write_M	I	CP0 写入使能信号
EXLclr_M	I	eret 信号
CP0Src_M	I	控制 CP0 数据信号
iflw_M	I	lw 信号
iflh_M	I	lh lhu 信号

信号名	方向	信号描述
`iflb_M`	I	`lb` `lbu` 信号
`ifsw_M`	I	`sw` 信号
`ifsh_M`	I	`sh` 信号
`ifsb_M`	I	`sb` 信号
`StoreType_M[1:0]`	I	M 级控制写入数据类型信号
`MuxData_M[2:0]`	I	M 级控制转发数据信号
`MemToReg_W`	I	W 级控制写入 `DM` 数据信号
`Link_W`	I	W 级写入 `PC + 8` 信号
`RegWrite_W`	I	W 级 `GRF` 写入使能信号
`ifmfhi_W`	I	W 级 `mfhi` 信号
`ifmflo_W`	I	W 级 `mflo` 信号
`DelaySlot_W`	I	延迟槽受害信号
`Instr_D[31:0]`	O	D 级指令
`Instr_E[31:0]`	O	E 级指令
`Instr_M[31:0]`	O	M 级指令
`Instr_W[31:0]`	O	W 级指令
`PCplus4_D[31:0]`	O	D 级 `PCplus4` 信号
`PCplus4_E[31:0]`	O	E 级 `PCplus4` 信号
`PCplus4_M[31:0]`	O	M 级 `PCplus4` 信号
`PCplus4_W[31:0]`	O	W 级 `PCplus4` 信号
`PCplus8_D[31:0]`	O	D 级 `PCplus8` 信号
`PCplus8_E[31:0]`	O	E 级 `PCplus8` 信号
`PCplus8_M[31:0]`	O	M 级 `PCplus8` 信号
`PCplus8_W[31:0]`	O	W 级 `PCplus8` 信号
`A3_E[4:0]`	O	E 级 `A3` 地址
`A3_M[4:0]`	O	M 级 `A3` 地址
`A3_W[4:0]`	O	W 级 `A3` 地址
`Busy_E`	O	E 级乘除槽繁忙信号
`PrAddr[31:0]`	O	`Timer` 写入地址
`PrWD[31:0]`	O	`Timer` 写入数据
`Addr[31:0]`	O	宏观 `PC`
`PrWe`	O	`Timer` 写入使能信号

DM

信号名	方向	信号描述
`clk`	I	时钟信号
`reset`	I	同步复位信号
`MemWrite`	I	写入使能信号
`Addr[31:0]`	I	数据存储器地址
`WriteData[31:0]`	I	写入数据
`PC[31:0]`	I	当前 `PC` 地址
`StoreType[1:0]`	I	数据类型信号
`DMOut[31:0]`	O	对应输出数据

序号	功能名称	功能描述
1	复位	当 `reset` 信号有效则将 `DM` 数据清零
2	写数据	当 `MemWrite` 有效且时钟上升沿时，将 `WriteData` 数据写入指定地址
3	读数据	从 `Addr` 读出数据

LoadData

信号名	方向	信号描述
`LoadType[2:0]`	I	控制数据类型信号
`DataIn[31:0]`	I	原始数据
`Offset[1:0]`	I	偏移量
`DataOut[31:0]`	O	根据类型定义后的数据

序号	功能名称	功能描述
1	转换数据	将数据转换成应有的数据类型

Ext

信号名	方向	信号描述
`ExtCtrl[1:0]`	I	控制扩展方式信号
`imm16[15:0]`	I	16 位立即数
`imm[31:0]`	O	扩展后的立即数

序号	功能名称	功能描述
1	扩展	按照 `ExtCtrl` 信号指定扩展数据和方式

GRF

信号名	方向	信号描述
`clk`	I	时钟信号
`reset`	I	同步复位信号
`RegWrite`	I	写入使能信号
`A1[4:0]`	I	指定 32 个寄存器中的一个，输出其中数据到 `RD1`
`A2[4:0]`	I	指定 32 个寄存器中的一个，输出其中数据到 `RD2`
`A3[4:0]`	I	指定 32 个寄存器中的一个，写入 `WriteData` 数据
`WriteData[31:0]`	I	输入数据
`PC[31:0]`	I	`PC` 地址
`RD1[31:0]`	O	`A1` 指定寄存器中的数据
`RD2[31:0]`	O	`A2` 指定寄存器中的数据

序号	功能名称	功能描述
1	复位	当复位信号有效时，寄存器清零
2	写数据	读出 `A1,A2` 的寄存器数据到 `RD1,RD2`
3	读数据	当 `RegWrite` 有效时且时钟上升沿时，将 `WriteData` 写入指定寄存器内

Hazard

信号名	方向	信号描述
`Instr_D[31:0]`	I	D 级指令信号
`Tuse_rs_D[2:0]`	I	D 级 `rs` 使用数据信号
`Tuse_rt_D[2:0]`	I	D 级 `rt` 使用数据信号
`ifmdu_D`	I	D 级乘除槽指令
`Instr_E[31:0]`	I	E 级指令信号
`Tnew_E[2:0]`	I	E 级数据完成信号
`A3_E[4:0]`	I	E 级 `A3` 地址
`start_E`	I	E 级触发乘除槽信号
`Busy_E`	I	E 级乘除槽繁忙信号
`Instr_M[31:0]`	I	M 级指令信号
`Tnew_M[2:0]`	I	M 级数据完成信号
`A3_M[4:0]`	I	M 级 `A3` 地址
`Instr_W[31:0]`	I	W 级指令信号
`Tnew_W[2:0]`	I	W 级数据完成信号
`A3_W[4:0]`	I	W 级 `A3` 地址
`ForwardSel_D1[4:0]`	O	D 级 `RD1` 转发信号
`ForwardSel_D2[4:0]`	O	D 级 `RD2` 转发信号
`ForwardSel_EA[4:0]`	O	E 级 `A` 转发信号
`ForwardSel_EB[4:0]`	O	E 级 `B` 转发信号
`ForwardSel_MD[4:0]`	O	M 级 `DM` 写入数据转发信号
`stall_PC`	O	暂停 `PC` 信号
`stall_FD`	O	暂停 `FD` 寄存器信号
`stall_DE`	O	暂停 `DE` 寄存器信号
`stall_EM`	O	暂停 `EM` 寄存器信号
`stall_MW`	O	暂停 `MW` 寄存器信号
`clr_FD`	O	清除 `FD` 寄存器信号
`clr_DE`	O	清除 `DE` 寄存器信号
`clr_EM`	O	清除 `EM` 寄存器信号
`clr_MW`	O	清除 `MW` 寄存器信号

Forward

信号名	方向	信号描述
ForwardSel_D1[4:0]	I	D 级 RD1 转发信号
ForwardSel_D2[4:0]	I	D 级 RD2 转发信号
ForwardSel_EA[4:0]	I	E 级 A 转发信号
ForwardSel_EB[4:0]	I	E 级 B 转发信号
ForwardSel_MD[4:0]	I	M 级 DM 写入数据转发信号
RD1_D[31:0]	I	D 级 RD1 数据
RD2_D[31:0]	I	D 级 RD2 数据
PCplus8_E[31:0]	I	E 级 PCplus8 数据
imm_E[31:0]	I	E 级 imm 数据
RD1_E[31:0]	I	E 级 RD1 数据
RD2_E[31:0]	I	E 级 RD2 数据
MuxData_M[2:0]	I	M 级数据选择信号
PCplus8_M[31:0]	I	M 级 PCplus8 数据
imm_M[31:0]	I	M 级 imm 数据
RD2_M[31:0]	I	M 级 RD2 数据
AluOut_M[31:0]	I	M 级 ALU 数据
HI_M[31:0]	I	M 级 HI 数据
LO_M[31:0]	I	M 级 LO 数据
MuxData_W[2:0]	I	W 级数据选择信号
PCplus8_W[31:0]	I	W 级 PCplus8 数据
imm_W[31:0]	I	W 级 imm 数据
AluOut_W[31:0]	I	W 级 ALU 数据
DMOut_W[31:0]	I	W 级 DM 数据
HI_W[31:0]	I	W 级 HI 数据
LO_W[31:0]	I	W 级 LO 数据
MF_RD1_D[31:0]	O	D 级 RD1 转发数据
MF_RD2_D[31:0]	O	D 级 RD1 转发数据
MF_A_E[31:0]	O	E 级 A 转发数据
MF_B_E[31:0]	O	E 级 B 转发数据
MF_RD2_M[31:0]	O	M 级 RD2 转发数据

序号	功能名称	功能描述
1	转发	转发前几级的数据

Comp

信号名	方向	信号描述
`zero`	I	比较零信号
`A[31:0]`	I	`RD1` 信号
`B[31:0]`	I	`RD2` 信号
`Comp_Out[2:0]`	O	比较后输出信号

序号	功能名称	功能描述
1	数据比较	若 $A > B$ 则 `Comp_Out = 100` 若 $A = B$ 则 `Comp_Out = 010` 若 $A < B$ 则 `Comp_Out = 001`

CP0

信号名	方向	信号描述
`clk`	I	时钟信号
`reset`	I	复位信号
`CP0Write`	I	写入使能信号
`EXLclr`	I	`eret` 信号
`PC[31:0]`	I	`PC` 值
`DataIn[31:0]`	I	写入数据
`Exccode[6:2]`	I	异常原因信号
`HWInt[5:0]`	I	中断信号
`Addr[4:0]`	I	寄存器地址
`DelaySlot`	I	延迟槽指令信号
`CP0Req`	O	异常中断请求
`epc[31:0]`	O	`EPC` 地址
`DataOut[31:0]`	O	`CP0` 读出数据

序号	功能名称	功能描述
1	复位	将寄存器清零及 `SR[0] = 1`
2	发生中断	发送中断请求
3	发生异常	发送异常请求
4	写入与读出	写入或读出寄存器的值

Bridge

信号名	方向	信号描述
PrRD0_In[31:0]	I	Timer0 读出数据
PrRD1_In[31:0]	I	Timer1 读出数据
PrAddr_In[31:0]	I	Timer 地址
PrWD_In[31:0]	I	Timer 写入数据
HWInt_In[15:10]	I	中断信号
PrWe_In	I	Timer 写入使能
PrRD_Out[31:0]	O	Timer 读出数据
PrAddr_Out[31:0]	O	Timer 地址
PrWD_Out[31:0]	O	Timer 写入数据
HWInt_Out[15:10]	O	中断信号
PrWe_Out0	O	Timer0 写入使能信号
PrWe_Out1	O	Timer1 写入使能信号

序号	功能名称	功能描述
1	传输	传输数据

CPU

信号名	方向	信号描述
clk	I	时钟信号
reset	I	复位信号
HWInt[15:10]	I	中断请求信号
PrRD[31:0]	I	Timer 读出数据
PrAddr[31:0]	O	Timer 地址
PrWD[31:0]	O	Timer 写入数据
Addr[31:0]	O	宏观 PC
PrWe	O	Timer 写入使能信号

Timer

信号名	方向	信号描述
clk	I	时钟信号
reset	I	复位信号
Addr[31:2]	I	地址
WE	I	写入使能信号
Din[31:0]	I	写入数据
Dout[31:0]	O	读出数据
IRQ	O	中断请求

序号	功能名称	功能描述
1	中断请求	产生中断请求

重要机制实现方法

1. 跳转

   利用 `Branch` 和 `Jump` 来判断下一个时钟沿写入的 `PC` 值

2. 转发

   利用 $Tuse_{rs}$ 或 $Tuse_{rt}$ $\geq$ 当前级数且 $Tnew \leq$ 当前级数，表示数据已经生成，可以直接转发

3. 暂停

   若 $Tuse_{rs} > Tnew$ ，则将处理器暂停，直到数据生成为止

二、测试方案

典型测试样例

汇编代码

```
.text 0x3000
lui $t0, 0x7fff
add $t1, $t0, $t0

.ktext 0x4180
mfc0 $k0, $14
addi $k0, $k0, 4
mtc0 $k0, $14
eret
```

机器码

```
.text
3c087fff
01084820
34090000
21290001
21290001
21290001
.ktext
401a7000
235a0004
409a7000
42000018
```

期望结果

```
@00003000: $ 8 <= 7fff0000
@00004180: $26 <= 00003004
@00004184: $26 <= 00003008
@00003008: $ 9 <= 00000000
@0000300c: $ 9 <= 00000001
@00003010: $ 9 <= 00000002
@00003014: $ 9 <= 00000003
```

汇编代码

```
.text
addi $ra, $0, 0x3005
jr $ra
ori $t0, 1
addi $t0, $t0, 1

.ktext 0x4180
mfc0 $k0, $14
addi $k0, $k0, 4
srl $k0, $k0, 2
sll $k0, $k0, 2
mtc0 $k0, $14
eret
```

机器码

```
.text
201f3005
03e00008
35080001
21080001
.ktext
401a7000
235a0004
001ad082
001ad080
409a7000
42000018
```

期望结果

```
@00003000: $31 <= 00003005
@00003008: $ 8 <= 00000001
@00004180: $26 <= 00003004
@00004184: $26 <= 00003008
@00004188: $26 <= 00000c02
@0000418c: $26 <= 00003008
@00003008: $ 8 <= 00000001
@0000300c: $ 8 <= 00000002
```

思考题

1. 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？）

   接口是两实体交换资料的介质，而硬件/软件接口就表示程序员能在不干预硬件的前提下就能利用软件来操作硬件来达到目的的程序。

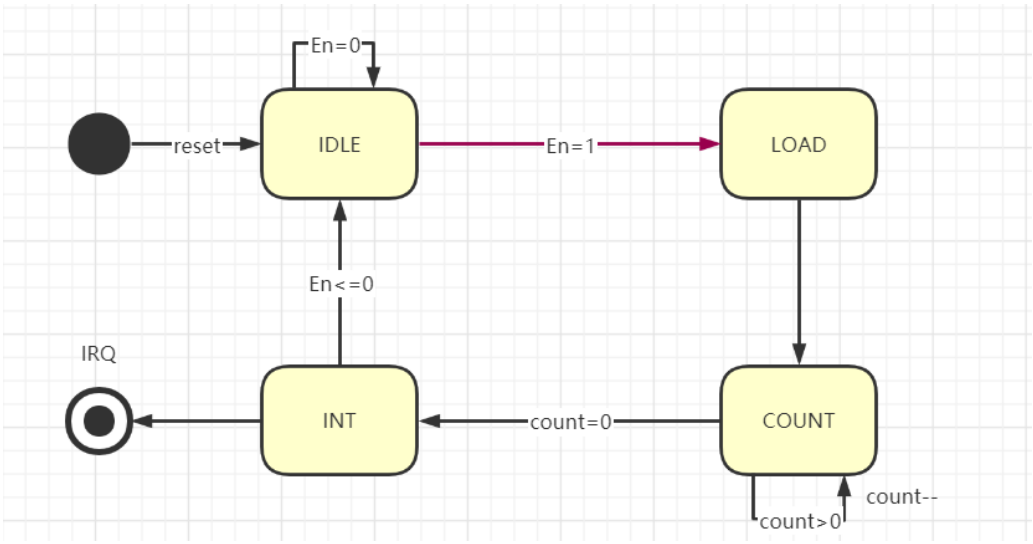
2. 在我们设计的流水线中，DM 处于 CPU 内部，请你考虑现代计算机中它的位置应该在何处。

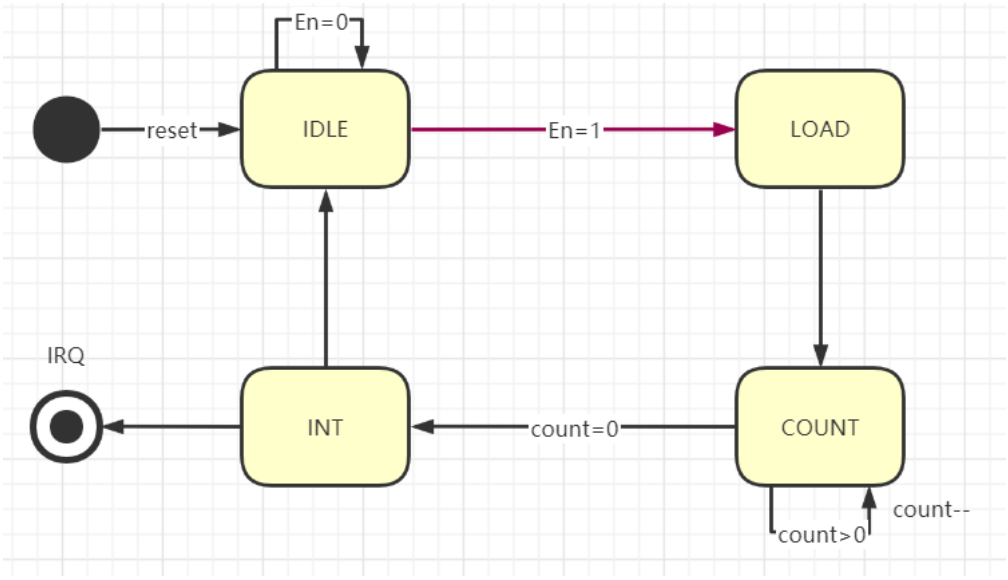
   设置在 CPU 外部

3. BE 部件对所有的外设都是必要的吗？

   不需要，因为外设需要传输一个字

4. 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





5. 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。）

```
.ktext 0x4180
mfc0 $k0, $14
addiu $k0, $k0, 4
srl $k0, $k0, 2
sll $k0, $k0, 2
mtc0 $k0, $14
eret
```

6. 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？

   键盘和鼠标在按下或者抬起的时候都会发送信号，键盘根据按下的情况并且使用组合利用组合逻辑来判断该发出什么中断操作，鼠标按下时进入缓冲区，抬起后再发送中断信号给处理器。