思考题

1. FPGA 是现场可编程门阵列，
   优点：灵活，有高层次内置模块，可以无限次编程，易于修改错误
   缺点：每次烧录后下电逻辑结构会丧失，面积较大速度较慢
2. UART 中断实现方案：
   核心目标：状态机进入对应状态时中断信号有效
   读取完毕寄存器后中断不再有效
   操作方法：修改 read_over 使之在被读寄存器后有效
   将 rs 引出作为中断信号
3. 看到 latch 没有和 FF（边沿触发器）一起出现在警告里就要小心
   是不是生成锁存器了….

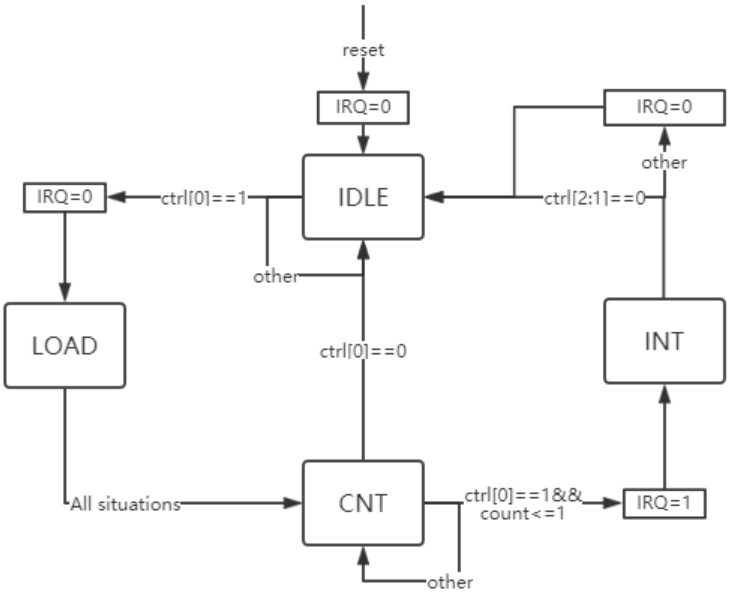
P78CPU 文档

~~（P6/7/8 工程化写玄学 bug 报告）~~

数据通路设计：

	地址或地址范围	备注
数据存储器	0x0000_0000 至 0x0000_1FFF	
指令存储器	0x0000_3000 至 0x0000_4FFF	
PC 初始值	0x0000_3000	
Exception Handler 入口地址	0x0000_4180	所有硬件的中断服务程序入口
设备寄存器地址	0x0000_7F00 至 0x0000_7F0B	定时器的 3 个寄存器
	0x0000_7F10 至 0x0000_7F2B	UART 的 7 个寄存器
	0x0000_7F2C 至 0x0000_7F33	64 位开关
	0x0000_7F34 至 0x0000_7F37	32 位 LED
	0x0000_7F38 至 0x0000_7F3F	9 位 8 段数码管
	0x0000_7F40 至 0x0000_7F43	8个用户按键

6. 计时器的状态转移图如下



- 上图只表明状态的转换，对于定时器内部寄存器值的改变还需要参考其余说明
- 上图中带有IRQ的方框表示在状态转换过程中IRQ值的变化，并不代表状态，箭头上的表达式代表状态转换的条件
- IRQ的值表示的状态机内部的值，在屏蔽中断状态下不论状态机内部IRQ为何值，对外表现出IRQ恒为0
- 外部写入的优先级大于状态机自身状态转换的优先级，换一种说法，在外部向计时器写入数据的周期中，计时器的状态寄存器不会发生变化，也不会进行count自减的操作

SR寄存器

IM[7:2]：6位中断屏蔽位，分别对应6个外部中断

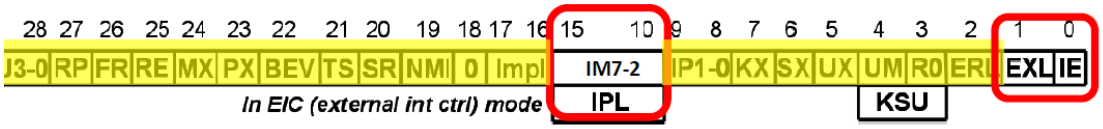
- 1-允许中断，0-禁止中断

IE：全局中断使能

- 1-允许中断；0-禁止中断

EXL：异常级

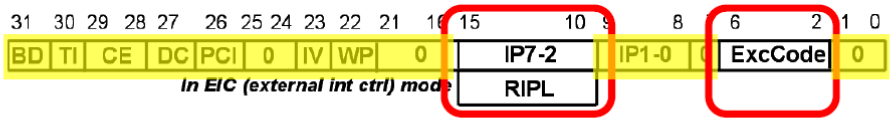
- 1-进入异常，不允许再中断；0-允许中断
- 注意：重入（在中断程序中仍然允许再次进行中断）需要OS的配合，重点是堆栈



CAUSE寄存器

- IP[7:2]：6位待决的中断位，分别对应6个外部中断
  - 记录当前哪些硬件中断正在有效
  - 1-有中断；0-无中断
- ExcCode[6:2]：异常编码，记录当前发生的是什么异常
  - 共计32种
  - 与课程相关的主要异常类型

ExcCode	助记符	描述
0	Int	中断
10	RI	不识别(非法)指令
12	Ov	算数指令导致的异常(如add)



正常各流水级部件

1. IFU

接口名称	类型	描述
clk	I	时钟信号
clr	I	复位信号
branch	I	跳转信号，为 1 时有效
npcout[31:0]	I	输入在跳转指令时需要跳转的地址
stall	I	暂停信号
instr	O	输出取出的指令
pc8	O	输出 pc+8 的值
intreq	I	中断信号

序号	功能名称	功能描述
1	复位	当 clr 信号有效时 pc 被置为 0x00003000
2.	取指令	取出 pc 对应地址里的指令
3	生成下一指令地址	若 branch 有效，下一指令地址为 npc 否则为 pc+4
4	stall 响应	若 stall 有效，锁定 pc 寄存器使其值不再改变
5.	intreq 响应	若 intreq 有效跳转至 npc（此时是 0x00004180）

2. grf

接口名	类型	描述
clk	I	时钟信号
reset	I	异步复位信号，1 为有效
we	I	写使能信号，1 为有效
a1[4:0]	I	RD1 端口输出的寄存器的编号
a2[4:0]	I	RD2 端口输出的寄存器的编号
a3[4:0]	I	要写入的寄存器的编号
wd	I	要写入选定寄存器的数据
rd1	O	输出 A1 端口选中寄存器的值(进行转发)
rd2	O	输出 A2 端口选中寄存器的值（进行转发）

功能描述

序号	功能名称	功能描述
1	复位	reset 信号有效时，所有寄存器的值置为 0
2	读寄存器	RD1,RD2 输出 A1,A2 选中寄存器的值
3	写寄存器	we 信号有效时 A3 选中的寄存器的值被写为 WD 的值

3. npc

接口名	类型	描述
npcop[1:0]	I	功能选择 00 依据 Imm 字段计算下一指令地址 01 依据指令后 26 位与 pc+4 高四位计算地址 02 输出 rd1 中的地址 03 0x00004180 04 eret 执行时需返回的地址
pc4	I	输入 pc+4

extout	I	输入 imm 进行 16 位扩展后的信号
rd1	I	输入 gpr 中 a1 选定寄存器的值
intsr	I	输入当前正在执行的指令
npcout	O	输出计算出的下一指令
intreq	I	中断信号，若有效 npcot 为 0x4180
epc	I	eret 跳转地址

功能描述

序号	功能名称	功能描述
1	实现立即数跳转	依据 Imm 字段计算下一指令地址
2	实现 j 类型跳转	依据指令后 26 位与 pc+4 高四位计算地址
3	实现寄存器跳转	输出 rd1 中的地址
4	实现 handler 跳转	输出 0x4180
5	实现 eret 跳转	输出 epc 的值（epc 值已经过显式转发）

4. ext

接口名	类型	描述
imm[15:0]	I	输入 16 位待扩展立即数
extout[31:0]	O	输出扩展后 32 位数据
extop[1:0]	I	功能选择 00 无符号扩展，01 有符号扩展，02Lui 扩展

功能说明

序号	功能名称	功能描述
1	无符号扩展	高位补 16 个 0
2	有符号扩展	高位补 16 个符号位
3	Lui 扩展	低位补 16 个 0

5. cmp

接口名	类型	描述
[31:0]a	I	输入待比较数 a
[31:0]	I	输入待比较数 b
[31:0]zero32	O	输出 zeroextent(a==b)
gtz	O	输出 a>0
gez	O	输出 a>=0
ltz	O	输出 a<0
lez	O	输出 a<=0

功能说明

序号	功能名称	功能描述
1	相等判断	若 a==b 时有效

6. alu

接口名	类型	描述
a[31:0]	I	输入 alu 的第一个操作数
b[31:0]	I	输入 alu 的第二个操作数
aluop[1:0]	I	alu 功能选择

		00 输出 b，01 为加运算，10 为减运算，11 为或运算，
aluout[31:0]	O	输出运算结果
overflow	O	输出有无溢出

功能描述

序号	功能名称	功能描述
1	加运算	输出 a1+a2
2	减运算	输出 a1-a2
3	或运算	输出 a1|a2
4	输出 b	为提高 lui 等指令的性能并降低复杂度而预留

7. dm

接口名	类型	描述
addr[31:0]	I	DM 输入的地址
wd[31:0]	I	DM 要写入的数据
clk	I	时钟信号
clr	I	异步复位信号
we	I	写使能信号，1 为有效，将 A 端口选中的字写为 INPUT 中数据
dmout[31:0]	O	输出 addr 端口选中的字的值
be	I	输入字节使能

功能描述

序号	功能名称	功能描述
1	复位	clr 信号有效时，所有主存的值置为 0
2	读主存	输出 addr 端口选中的字的值
3	写主存	we 有效时将 addr 端口选中的字写为 wd 中数据

异常处理单元

1. Exc_i

接口名	方向	描述
instr_d	I	d 级指令
pc8_i	I	I 级指令的 pc8 值
bd_i	O	branch delayed 信号，若为延迟槽指令则为 1
exccode_i	O	传递异常代码

功能说明

序号	功能名称	描述
1	判断取指异常	若 pc 越界或不对齐报 adel 异常
2	延迟槽指令判断	若为延迟槽指令则置 1

2. Exc_d

接口名	方向	描述
instr_d	I	输入 d 级指令
exccode_i	I	输入 i 级传递的 exccode
bd_i	I	输入是否延迟槽指令
exccode_d	O	输出 exccode
bd_d	O	输出 bd

功能说明

序号	功能名称	描述
1	检查不识别指令	若指令未被识别则报 ri 异常
2	传递前级异常	若本级无异常则传递前级 exccode

3. exc_e

接口名	方向	描述
instr_e	I	输入 e 级指令
overflow	I	输入 overflow 判断结果
bd_d	I	输入前级 bd
exccode_d	I	输入前级 exccode
bd_e	O	输出 bd
exccode_e	O	输出 exccode

功能说明

序号	功能名称	描述
1	检查算数溢出	若 add sub 算数溢出报 ov 异常
2	传递前级异常	若本级无异常则传递前级 exccode

4. exc_m

接口名	方向	描述
instr_m	I	输入 m 级指令
ao_m	I	输入 m 级地址总线
exccode_e	I	输入前级 exccode
bd_e	I	输入前级 bd
exccode_m	O	输出 exccode
bd_m	O	输出 bd

功能说明

序号	功能名称	描述
1	判断读/写异常	若读写地址不对齐/地址越界/写不该写的地址 报对应的异常
2	传递前级异常	若本级无异常则传递前级 exccode

协处理器单元 0

接口名	方向	描述
clk	I	时钟信号
clr	I	复位信号
a1	I	读寄存器编号
a2	I	写寄存器编号
wdbus	I	写入数据
pc8_m	I	写入 m 级异常/中断受害指令地址
bdin	I	写入 m 级异常/中断受害指令是否为延迟槽指令
hwint	I	输入 6 个外部设备中断信号
exccodein	I	输入 m 级异常/中断编号
we	I	CP0 寄存器写使能
exlclr	I	关中断使能
cp0_rd	O	cp0 读出数据
cp0_epc	O	epc 寄存器的值

intreq	O	中断信号

功能说明

序号	功能名称	描述
1	复位	若 clr 信号有效实施复位操作
2	外部中断响应	若外部中断到来且全局允许中断且不在中断时 保存 epc 值为 pc_m(pc_m-4 若为延迟槽) 保存 cause 寄存器的值 exl 置位为 1 cause exccode 段置位 00000
3	异常响应	若 m 级指令异常且全局允许中断且不在中断时 保存 epc 值为 pc_m(pc_m-4 若为延迟槽) 保存 cause 寄存器的值 exl 置位为 1
4	关中断信号	若有效 exl 置位 0
5	写 CP0 寄存器	若写使能有效，写入对应编号的寄存器
6	读 CP0 寄存器	输出对应编号的寄存器值

Bridge 设计

接口名	方向	描述
we	I	写使能信号
addrbus	I	地址总线
wdbus	I	数据总线
bridge_rd	O	读出数据
hwint	O	输出中断位
t_addr	O	输出地址
t_wd	O	写入数据
t_we	O	写使能
t_rd0	I	定时器读数据
t_int0	I	定时器中断
uart_rd	O	uard 读数据
uart_int	I	uart 中断
stb	O	uart 选中信号
switch_rd	I	开关输出
switch_int	O	开关中断
led_rd	I	led 读入
tube8_rd	I	8 段数码管读入
userkey_rd	I	用户按键读入
userkey_int	O	用户按键中断

新增支持半字/字节模块

M 级 dmdecode

接口名	方向	描述
wordmode	I	输入 BE 的译码模式
addr	I	输入当前要读取的地址

be	O	输出 BE 译码结果

功能说明

编号	功能	描述
1	BE 译码	根据输入输出 BE 的值

W 级 dmcut

接口名	方向	描述
dmout	I	输入完整 DM 输出结果
ao_w	I	输入当前地址
dr_w	O	输出需要的结果
wordmode	I	输入字节/半字/字模式

功能说明

编号	功能	描述
1	无符号半字	输出无符号扩展的指定半字
2	有符号半字	输出有符号扩展的指定半字
3	无符号字节	输出无符号扩展的指定字节
4	有符号字节	输出有符号扩展的指定字节
5	整字输出	输出完整的字

新增各个驱动模块定义

miniuart

接口名	方向	描述
ADD_I	I	寄存器地址
DAT_I	I	地址
DAT_O	O	读输出输出
STB_I	I	选中信号
WE_I	I	写使能
CLK_I	I	时钟信号
RST_I	I	复位信号
RXD	I	输入数据线
TXD	O	输出数据线
uartint	O	中断信号

Switch

接口名	方向	描述
dip_switch0,	I	各个开关电平输入
dip_switch1		
dip_switch2,		
dip_switch3,		
dip_switch4		
dip_switch5,		
dip_switch6,		
dip_switch7,		
clk	I	时钟信号
reset	I	复位

switch_rd	O	读出数据
switch_int	O	中断信号

Userkeydriver

接口名	方向	描述
clk	I	时钟信号
clr	I	清零信号
user_key	I	用户按键输入
userkey_rd	O	用户按键读取
userkey_int	O	中断信号

Led

接口名	方向	描述
clk	I	时钟信号
clr	I	清零信号
led_light	I	led 输出
led_rd	O	led 读取
t_wd	I	写入数据
t_we	I	写使能信号
addr	I	地址

接口名	方向	描述
digital_tube0	O	数码管显示信号
digital_tube1	O	
digital_tube2	O	
sel0	O	片选信号 0
sel1	O	片选信号 1
sel2	O	片选信号 2
clk	I	时钟
clr	I	清零
t_addr	I	地址
t_we	I	写使能
t_wd	I	写入数据
octled_rd	O	数码管读出数据

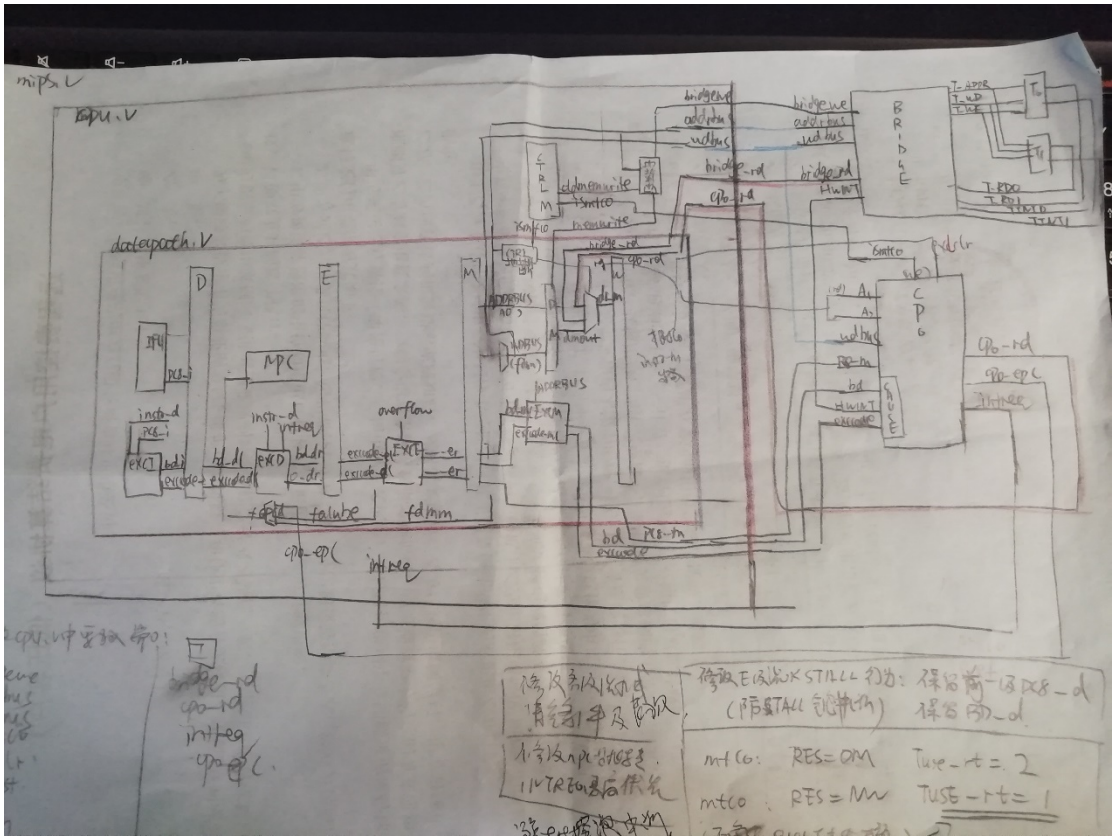
各级流水线寄存器

```
module Dpipe (clk,clr,en,npcout,iseret,
              instr_i,pc8_i,bd_i,exccode_i,
              instr_d,pc8_d,bd_d,exccode_d);
```

```
3  module Epipe(clk,clr,stall,
4  res_d,instr_d,a1_d,a2_d,a3_d,v1_d,v2_d,oth_d,pc8_d,bd_d,exccode_d,
5  res_e,instr_e,a1_e,a2_e,a3_e,v1_e,v2_e,oth_e,pc8_e,bd_e,exccode_e);
```

```
3  module Mpipe(clk,clr,
4  res_e,instr_e,a2_e,a3_e,v2_e,ao_e,pc8_e,bd_e,exccode_e,
5  res_m,instr_m,a2_m,a3_m,v2_m,ao_m,pc8_m,bd_m,exccode_m);
```

DATAPATH 及 CP0，BRIDGE,TIMER 缩略图



控制器设计：

控制器部分基本沿用 P5 设计

1. 常规控制器设计

线网类型变量：

```
assign isbeq=(op==`beq);
assign isbne=(op==`bne);
assign ismfc0=(op==`cop0&&rs==`mfc0_rs);
assign ismtc0=(op==`cop0&&rs==`mtc0_rs);
assign wordmode=(op==`lbu)?`wm_bu:
                (op==`lh||op==`sh)?`wm_hs:
                (op==`lhu)?`wm_hu:
                (op==`lb||op==`sb)?`wm_bs:
                `wm_wd;

assign  mdop=((op==`special)&&funct==`divu_funct&&instr[15:6]==0)?`md_divu:
             ((op==`special)&&funct==`div_funct&&instr[15:6]==0)?`md_div:
             ((op==`special)&&funct==`multu_funct&&instr[15:6]==0)?`md_multu:
             `md_mult;
assign usemd=(op==`special&&instr[20:6]==0&&(funct==`mthi_funct||funct==`mtlo_funct)
            ||op==`special&&instr[25:16]==0&&instr[10:6]==0&&(funct==`mfhi_funct||funct==`mflo_funct)
            ||(op==`special&&funct==`divu_funct&&instr[15:6]==0)
            ||(op==`special&&funct==`div_funct&&instr[15:6]==0)
            ||(op==`special&&funct==`multu_funct&&instr[15:6]==0)
            ||(op==`special&&funct==`mult_funct&&instr[15:6]==0));
assign start=(op==`special&&funct==`divu_funct&&instr[15:6]==0)
            ||(op==`special&&funct==`div_funct&&instr[15:6]==0)
            ||(op==`special&&funct==`multu_funct&&instr[15:6]==0)
            ||(op==`special&&funct==`mult_funct&&instr[15:6]==0);
```

Addu add subu sub and or xor nor

```
if(op==`special &&shamt==0&&(funct==`addu_funct||funct==`add_funct||funct==`subu_funct||funct==`sub_funct
                  ||funct==`and_funct||funct==`or_funct||funct==`xor_funct||funct==`nor_funct))
   begin//addu add subu sub and or xor nor
   debug=1;
      branch<=0; npcop<=0; extop<=0; regdst<=`regdst_rd; otherctrl<=0;
      alusrc<=`alusrc_rd2; memwrite<=0; memtoreg<=`memtoreg_ao; regwrite<=1;
      if (funct==`addu_funct||funct==`add_funct)
         aluop<=`alu_add;
      else if(funct==`subu_funct||funct==`sub_funct)
         aluop<=`alu_sub;
      else if(funct==`and_funct)
         aluop<=`alu_and;
      else if(funct==`or_funct)
         aluop<=`alu_or;
      else if(funct==`xor_funct)
         aluop<=`alu_xor;
      else if(funct==`nor_funct)
         aluop<=`alu_nor;
      else
         aluop<=`alu_debug;
   end
```

ori andi xori

jr

addiu addi

```
   else if(op==`ori||op==`andi||op==`xori)
      begin//ori andi xori
         debug=2;
         branch<=0; npcop<=0; extop<=`ext_unsign; regdst<=`regdst_rt; otherctrl<=`oth_ext;
         alusrc<=`alusrc_oth; memwrite<=0; memtoreg<=`memtoreg_ao; regwrite<=1;
         if(op==`ori)
               aluop<=`alu_or;
         else if(op==`andi)
               aluop<=`alu_and;
         else if(op==`xori)
               aluop<=`alu_xor;

      end
   else if(op==`special&&shamt==0&&funct==`jr_funct&&instr[20:6]==0)
      begin//jr
         branch<=1; npcop<=`npc_reg; extop<=0; regdst<=0; otherctrl<=0; aluop<=0;
         alusrc=0; memwrite<=0; memtoreg<=0; regwrite<=0;
      end

   else if(op==`addiu||op==`addi)
      begin//addiu addi
         branch<=0; npcop<=0; extop<=`ext_sign; regdst<=`regdst_rt; otherctrl<=`oth_ext; aluop<=`alu_add;
         alusrc<=`alusrc_oth; memwrite<=0; memtoreg<=`memtoreg_ao; regwrite<=1;
      end
```

lw lh lhu lb lbu

sw sb sh

sll sra srl

sllv srlv srav

```
   else if(op==`lw||op==`lh||op==`lhu||op==`lb||op==`lbu)
      begin//lw lh lhu lb lbu
         branch<=0; npcop<=0; extop<=`ext_sign; regdst<=`regdst_rt; otherctrl<=`oth_ext; aluop<=`alu_add;
         alusrc<=`alusrc_oth; memwrite<=0; memtoreg<=`memtoreg_dr; regwrite<=1;
      end
   else if(op==`sw||op==`sb||op==`sh)
      begin//sw sb sh
         branch<=0; npcop<=0; extop<=`ext_sign; regdst<=0; otherctrl<=`oth_ext; aluop<=`alu_add;
         alusrc<=`alusrc_oth; memwrite<=1; memtoreg<=0; regwrite<=0;
      end
   else if(op==`special&&rs==0&&(funct==`sll_funct||funct==`sra_funct||funct==`srl_funct))
      begin//sll sra srl
         branch<=0; npcop<=0; extop<=0; regdst<=`regdst_rd; otherctrl<=0;
         alusrc<=`alusrc_rd2; memwrite<=0; memtoreg<=`memtoreg_ao; regwrite<=1;
            if(funct==`sll_funct) aluop<=`alu_sll;
            else if(funct==`srl_funct) aluop<=`alu_srl;
            else if(funct==`sra_funct) aluop<=`alu_sra;
      end
   else if(op==`special&&shamt==0&&(funct==`sllv_funct||funct==`srlv_funct||funct==`srav_funct))
      begin//sllv srlv srav
         branch<=0; npcop<=0; extop<=0; regdst<=`regdst_rd; otherctrl<=0;
         alusrc<=`alusrc_rd2; memwrite<=0; memtoreg<=`memtoreg_ao; regwrite<=1;
            if (funct==`sllv_funct) aluop<=`alu_sllv;
            else if(funct==`srlv_funct) aluop<=`alu_srlv;
            else if(funct==`srav_funct) aluop<=`alu_srav;
      end
```

slt sltu

slti sltiu

beq bgtz bgez blez bltz bne

lui

```
else if(op==`special&&shamt==0&&(funct==`slt_funct||funct==`sltu_funct))
   begin//slt sltu
      branch<=0; npcop<=0; extop<=0; regdst<=`regdst_rd; otherctrl<=0;
      alusrc<=`alusrc_rd2; memwrite<=0; memtoreg<=`memtoreg_ao; regwrite<=1;
      if(funct==`slt_funct) aluop<=`alu_slt;
      else if(funct==`sltu_funct) aluop<=`alu_sltu;
   end
else if(op==`slti||op==`sltiu)
   begin//slti sltiu
      branch<=0; npcop<=0; extop<=`ext_sign; regdst<=`regdst_rt; otherctrl<=`oth_ext;
      alusrc<=`alusrc_oth; memwrite<=0; memtoreg<=`memtoreg_ao; regwrite<=1;
      if(op==`slti)      aluop<=`alu_slt;
      else if(op==`sltiu) aluop<=`alu_sltu;
   end
else if(op==`beq||op==`bne
   ||((op==`bgtz||op==`blez)&&rt==0)
   ||op==`bgezbltz)
   begin//beq bgtz blez bgez bltz bne
      branch<=1; npcop<=`npc_16; extop<=`ext_sign; regdst<=0; otherctrl<=0; aluop<=0;
      alusrc<=0; memwrite<=0; memtoreg<=0; regwrite<=0;
   end
else if(op==`lui)
   begin//lui
      branch<=0; npcop<=0; extop<=`ext_lui; regdst<=`regdst_rt; otherctrl<=`oth_ext; aluop<=`alu_oth;
      alusrc<=`alusrc_oth; memwrite<=0; memtoreg<=`memtoreg_ao; regwrite<=1;
   end
```

Jal j jalr eret mfc0

```
else if(op==`jal)
   begin//jal
      branch<=1; npcop<=`npc_26; extop<=0; regdst<=`regdst_31; otherctrl<=0; aluop<=0;
      alusrc<=0; memwrite<=0; memtoreg<=`memtoreg_pc; regwrite<=1;
   end
else if (op==`j)
   begin//j
      branch<=1; npcop<=`npc_26; extop<=0; regdst<=0; otherctrl<=0; aluop<=0;
      alusrc=0; memwrite<=0; memtoreg<=0; regwrite<=0;
   end
else if(op==`special&&funct==`jalr_funct&&shamt==0)
   begin//jalr
      branch<=1; npcop<=`npc_reg; extop<=0; regdst<=`regdst_rd; otherctrl<=0; aluop<=0;
      alusrc<=0; memwrite<=0; memtoreg<=`memtoreg_pc; regwrite<=1;
   end
else if(instr==`eret)
   begin//eret
      branch<=1; npcop<=`npc_epc; extop<=0; regdst<=0; otherctrl<=0; aluop<=0;
      alusrc<=0; memwrite<=0; memtoreg<=0; regwrite<=0;
   end
else if(op==`cop0&&rs==`mfc0_rs&&instr[10:0]==0)
   begin//mfc0
      branch<=0; npcop<=0; extop<=0; regdst<=`regdst_rt; otherctrl<=0; aluop<=0;
      alusrc<=0; memwrite<=0; memtoreg<=`memtoreg_dr; regwrite<=1;
   end
```

Mtc0 mult multu div divu mfhi mflo mthi mtlo

```
else if(op==`cop0&&rs==`mtc0_rs&&instr[10:0]==0)
   begin//mtc0
      branch<=0; npcop<=0; extop<=0; regdst<=0; otherctrl<=0; aluop<=0;
      alusrc<=0; memwrite<=0; memtoreg<=0; regwrite<=0;
   end
else if(op==`special&&instr[15:6]==0&&(funct==`mult_funct||funct==`multu_funct||funct==`div_funct||funct==`divu_funct))
   begin//mult multu div divu
      branch<=0; npcop<=0; extop<=0; regdst<=0; otherctrl<=0; aluop<=0;
      alusrc<=`alusrc_rd2; memwrite<=0; memtoreg<=0; regwrite<=0;
   end
else if(op==`special &&(funct==`mfhi_funct||funct==`mflo_funct) )
   begin//mfhi mflo
   debug=15;
      branch<=0; npcop<=0; extop<=0; regdst<=`regdst_rd; otherctrl<=0; aluop<=0;
      alusrc<=`alusrc_rd2; memwrite<=0; memtoreg<=`memtoreg_ao; regwrite<=1;
   end
else if(op==`special &&(funct==`mthi_funct || funct ==`mtlo_funct))
   begin//mtli mtlo
   debug=16;
      branch<=0; npcop<=0; extop<=0; regdst<=0; otherctrl<=0; aluop<=0;
      alusrc<=0; memwrite<=0; memtoreg<=0; regwrite<=0;
   end
```

AT 控制器

Tuse_rs

```
if(
   (op==`special&&(
                    funct==`addu_funct||funct==`subu_funct||funct==`add_funct||funct==`sub_funct
                  ||funct==`and_funct||funct==`or_funct||funct==`xor_funct||funct==`nor_funct
                  ||funct==`sllv_funct||funct==`srlv_funct||funct==`srav_funct
                  ||funct==`slt_funct||funct==`sltu_funct
                  ||funct==`mult_funct||funct==`multu_funct||funct==`div_funct||funct==`divu_funct
                  ||funct==`mtlo_funct||funct==`mthi_funct
                  ))
    ||op==`ori||op==`andi||op==`addiu||op==`addi||op==`xori
    ||op==`lw||op==`lh||op==`lb||op==`lbu||op==`lhu
    ||op==`sw||op==`sb||op==`sh
    ||op==`slti||op==`sltiu
    )
   tuse_rs<=1;
else if(op==`special&&(funct==`jr_funct||funct==`jalr_funct )
       || op==`beq||op==`bne||op==`bgtz||op==`blez||op==`bgezbltz)
   tuse_rs<=0;
else
   tuse_rs<=3;
''
```

Tuse_rt

```
   if(
      (op==`special&&(
                    funct==`addu_funct||funct==`subu_funct||funct==`add_funct||funct==`sub_funct
                    ||funct==`and_funct||funct==`or_funct||funct==`xor_funct||funct==`nor_funct
                    ||funct==`sll_funct||funct==`srl_funct||funct==`sra_funct
                    ||funct==`sllv_funct||funct==`srlv_funct||funct==`srav_funct
                    ||funct==`slt_funct||funct==`sltu_funct
                    ||funct==`mult_funct||funct==`multu_funct||funct==`div_funct||funct==`divu_funct
                    ))
      ||(op==`cop0&&rs==`mtc0_rs))
      tuse_rt<=1;
   else if(op==`beq||op==`bne)
      tuse_rt<=0;
   else if(op==`sw||op==`sb||op==`sh)
      tuse_rt<=2;
   else
      tuse_rt<=3;
   ''
```

Res

```
  if(
     (op==`special&&(
                    funct==`addu_funct||funct==`subu_funct||funct==`add_funct||funct==`sub_funct
                    ||funct==`and_funct||funct==`or_funct||funct==`xor_funct||funct==`nor_funct
                    ||funct==`sll_funct||funct==`srl_funct||funct==`sra_funct
                    ||funct==`sllv_funct||funct==`srlv_funct||funct==`srav_funct
                    ||funct==`slt_funct||funct==`sltu_funct
                    ||funct==`mflo_funct||funct==`mfhi_funct
                    ))
    ||op==`ori||op==`andi||op==`addiu||op==`addi||op==`xori||op==`slti||op==`sltiu)
     res_d<=`alu;
  else if(op==`jal||op==`special&&funct==`jalr_funct)
     res_d<=`pc;
  else if(op==`lw||op==`lb||op==`lbu||op==`lh||op==`lhu
         ||(op==`cop0&&rs==`mfc0_rs))
     res_d<=`dm;
  else if(op==`lui)
     res_d<=`other;
  else
     res_d<=`nw;
```

STALL 控制器：无任何修改

```
assign stall_rs01e=(tuse_rs==0)&&(res_e==`alu)&&(a1_d==a3_e)&&(a1_d!=0);
assign stall_rs02e=(tuse_rs==0)&&(res_e==`dm)&&(a1_d==a3_e)&&(a1_d!=0);
assign stall_rs12e=(tuse_rs==1)&&(res_e==`dm)&&(a1_d==a3_e)&&(a1_d!=0);
assign stall_rs01m=(tuse_rs==0)&&(res_m==`dm)&&(a1_d==a3_m)&&(a1_d!=0);
assign stall_rs=stall_rs01e|stall_rs02e|stall_rs12e|stall_rs01m;

assign stall_rt01e=(tuse_rt==0)&&(res_e==`alu)&&(a2_d==a3_e)&&(a2_d!=0);
assign stall_rt02e=(tuse_rt==0)&&(res_e==`dm)&&(a2_d==a3_e)&&(a2_d!=0);
assign stall_rt12e=(tuse_rt==1)&&(res_e==`dm)&&(a2_d==a3_e)&&(a2_d!=0);
assign stall_rt01m=(tuse_rt==0)&&(res_m==`dm)&&(a2_d==a3_m)&&(a2_d!=0);
assign stall_rt=stall_rt01e|stall_rt02e|stall_rt12e|stall_rt01m;

assign stall=stall_rs|stall_rt;
```

转发控制器：无任何修改

```
31  assign fv1dctrl=(res_e==`pc && a1_d==a3_e && a1_d!=0)?`fd_pc8_e:
32                  (res_e==`other && a1_d==a3_e &&a1_d!=0)?`fd_oth_e:
33                  (res_m==`alu && a1_d==a3_m && a1_d!=0)?`fd_ao_m:
34                  (res_m==`pc && a1_d==a3_m && a1_d!=0)?`fd_pc8_m:
35                  `fd_rd;
36  assign fv2dctrl=(res_e==`pc && a2_d==a3_e && a2_d!=0)?`fd_pc8_e:
37                  (res_e==`other && a2_d==a3_e &&a2_d!=0)?`fd_oth_e:
38                  (res_m==`alu && a2_d==a3_m && a2_d!=0)?`fd_ao_m:
39                  (res_m==`pc && a2_d==a3_m && a2_d!=0)?`fd_pc8_m:
40                  `fd_rd;
41  assign faluaectrl=(res_m==`pc &&a1_e==a3_m&&a1_e!=0)?`falue_pc8_m:
42                    (res_m==`alu &&a1_e==a3_m&&a1_e!=0)?`falue_ao_m:
43                    (res_w!=`nw && a1_e==a3_w&&a1_e!=0)?`falue_wd_w:
44                    `falue_v;
45  assign falubectrl=(res_m==`pc &&a2_e==a3_m&&a2_e!=0)?`falue_pc8_m:
46                    (res_m==`alu &&a2_e==a3_m&&a2_e!=0)?`falue_ao_m:
47                    (res_w!=`nw && a2_e==a3_w&&a2_e!=0)?`falue_wd_w:
48                    `falue_v;
49  assign fdmmctrl=(res_w!=`nw && a2_m==a3_w &&a2_m!=0)?1:0;
```

（epc 的转发在 datapath 中独立完成，不受 forward 控制器的控制）

（乘除模块的暂停，中止，回滚在 datapath 中完成）

测试程序

1.计算器

```
loop:
    lw $s0,0x7f2c($0)
    lw $s1,0x7f30($0)
    lw $s2,0x7f40($0)

    ori $s3,$0,1
    bne $s3,$s2,else1
    nop
        addu $s4,$s0,$s1
        j end
        nop
    else1:
    ori $s3,$0,2
    bne $s3,$s2,else2
    nop
```

```
        subu $s4,$s0,$s1
        j end
        nop
    else2:
    ori $s3,$0,4
    bne $s3,$s2,else3
    nop
        or $s4,$s0,$s1
        j end
        nop
    else3:
    ori $s3,$0,8
    bne $s3,$s2,else4
    nop
        and $s4,$s0,$s1
        j end
        nop
    else4:
    ori $s3,$0,16
    bne $s3,$s2,else5
    nop
        xor $s4,$s0,$s1
        j end
        nop
    else5:
    ori $s3,$0,32
    bne $s3,$s2,else6
    nop
        nor $s4,$s0,$s1
        j end
        nop
    else6:
    ori $s3,$0,64
    bne $s3,$s2,else7
    nop
        slt $s4,$s0,$s1
        j end
        nop
    else7:
    ori $s3,$0,128
    bne $s3,$s2,end
    nop
        sllv  $s4,$s0,$s1
        j end
```

```
        nop
    end:
    sw $s4,0x7f38($0)
    sw $s4,0x7f34($0)
    j loop
    nop
```

2. 倒计时

```
.text
1.      ori $s0,$0,11
2.      li $s1,20000000
3.      sw $s0,0x7f00($0)#CTRL
4.      sw $s1,0x7f04($0)#PRESET
5.  loop:
6.      lw $s2,0x7f2c($0)#switch
7.      beq $s2,$s3,end#s3 switch old value
8.      nop
9.          move $s3,$s2
10.         move $s4,$s3#$s4 counting
11.     end:
12.     sw $s4,0x7f38($0)
13.     j loop
14.     nop
15..ktext 0x4180
16.     blez $s4,else
17.     nop
18.         addiu $s4,$s4,-1
19.         j endif
20.         nop
21.     else:
22.         ori $s4,$0,0
23.     endif:
24.     eret
25.     nop
26.
27.
```

3.UART

```
1.  .text
2.  #k0 is start k1 is end
3.  ori $a1,$0,32
4.  start:
5.  slt $t7,$k0,$k1
6.  beq $t7,$0,else
7.  nop
```

```
8.      loop:
9.          lw $a0,0x7f20($0)
10.         andi $a0,$a0,32
11.     bne $a0,$a1,loop
12.     nop
13.     lw $a3,0($k0)
14.     addiu $k0,$k0,4
15.     sw $a3,0x7f10($0)
16.     sw $k0,0x7f38($0)
17.
18.else:
19.j start
20.nop
21.
22..ktext 0x4180
23.lw $a2,0x7f10($0)
24.sw $a2,0($k1)
25.addiu $k1,$k1,4
26.eret
27.
```