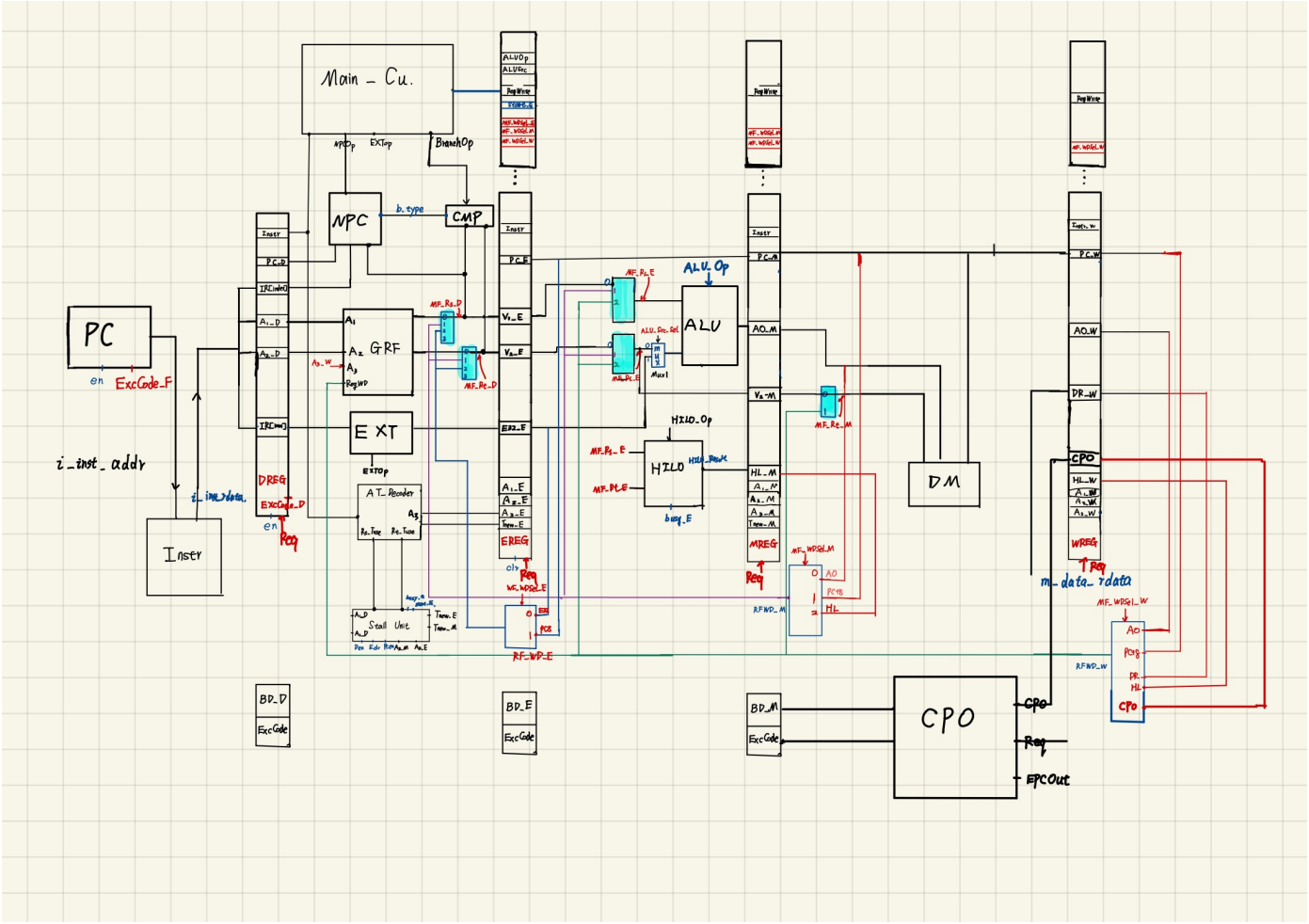
计算机组成P7实验报告

设计文档

一、整体架构



本处理器支持的指令集为

{nop, add, sub, and, or, slt, sltu, lui,addi, andi, ori,lb, lh, lw, sb, sh, sw,mult, multu, div, divu, mfhi, mflo,mthi, mtlo,beq, bne, jal, jr, mfc0, mtc0, eret, syscall}

本处理器支持的中断异常处理包括：

异常与中断码	助记符与名称	指令与指令类型	描述
0	`Int`（外部中断）	所有指令	中断请求，来源于计时器与外部中断。
4	`AdEL`（取指异常）	所有指令	PC 地址未字对齐。
PC 地址超过 `0x3000 ~ 0x6ffc`。			
`AdEL`（取数异常）	`lw`	取数地址未与 4 字节对齐。	
`lh`	取数地址未与 2 字节对齐。		
`lh`, `lb`	取 Timer 寄存器的值。		
load 型指令	计算地址时加法溢出。		
load 型指令	取数地址超出 DM、Timer0、Timer1、中断发生器的范围。		
5	`AdES`（存数异常）	`sw`	存数地址未 4 字节对齐。
`sh`	存数地址未 2 字节对齐。		
`sh`, `sb`	存 Timer 寄存器的值。		
store 型指令	计算地址加法溢出。		
store 型指令	向计时器的 Count 寄存器存值。		
store 型指令	存数地址超出 DM、Timer0、Timer1、中断发生器的范围。		
8	`Syscall`（系统调用）	`syscall`	系统调用。
10	`RI`（未知指令）	-	未知的指令码。
12	`Ov`（溢出异常）	`add`, `addi`, `sub`	算术溢出。

二、模块规格

1.F_PC模块

输入输出端口设计：

信号名	方向	描述
clk	I	时钟信号
reset	I	异步复位信号，将PC寄存器清零。1：复位；0：无效。
en	I	使能信号，用于阻塞
NPC[31:0]	I	32位输入信号，下一条指令的地址
PC[31:0]	O	32位输出信号，当前指令的地址

2.D_NPC模块

信号名	方向	描述
PC_F [31:0]	I	32位输入信号，F阶段指令的地址
PC_D[31:0]	I	32位输入信号，D阶段指令的地址
NPCOp[2:0]	I	下一个PC的选择信号
BranchSignal	I	1位输入信号，表示CMP模块比较结果
offset [31:0]	I	32位输入信号，代表地址偏移量
Instr_index [25:0]	I	26位输入信号，代表j/jal指令的26位地址索引
RegData [31:0]	I	32位输入信号，代表jr指令目标寄存器的值
NPC [31:0]	O	32位输出信号，下一个PC的地址

3.D_CMP模块

输入输出端口设计：

信号名	方向	作用
[31:0] MF_Rs_D	input	32位输入信号，代表branch类型指令需要的Rs数据
[31:0] MF_Rt_D	input	32位输入信号，代表branch类型指令需要的Rt数据
[2:0] BranchOp	input	3位输入信号，选择Branch类型中不同的指令
BranchSignal	output	1位输入信号，代表跳转条件是否成立

4.D_EXT模块

信号名	方向	描述
Imm [15:0]	I	16位输入信号，表示待进行拓展的立即数
EXTOp [2:0]	I	3位输入信号，代表EXT选择的操作
EXTOut [31:0]	O	32位输出信号，表示拓展后的立即数

5.D_GRF模块

信号名	方向	描述
clk	I	时钟信号
reset	I	异步复位信号，将32个寄存器的值全部清零
RegWrite	I	写使能信号，高电平时向A3寄存器写入数据WD3；低电平时无效
A1 [4:0]	I	5位输入信号，选择32个寄存器中的一个并将其数据输出至RD1端口
A2 [4:0]	I	5位输入信号，选择32个寄存器中的一个并将其数据输出至RD2端口
A3 [4:0]	I	5位输入信号，选择32个寄存器中的一个作为数据写入的目标寄存器
GRF_WD [31:0]	I	32位输入数据，当WE3有效时向GRF中写入数据
RF_RD1 [31:0]	O	32位输出信号，输出A1中选择的寄存器中的数据
RF_RD2 [31:0]	O	32位输出信号，输出A2中选择的寄存器中的数据

6.E_ALU模块

信号名	方向	描述
Src_A [31:0]	I	32位输入信号，待ALU处理的数据之一
Src_B [31:0]	I	32位输入信号，待ALU处理的数据之一
ALUOp [3:0]	I	3位输入选择信号，选择ALU需要进行的操作
ALUOut[31:0]	O	32位输出信号，ALU运算的输出结果

7._HILO模块：

信号名称	方向	功能
clk	I	时钟信号
reset	I	复位信号，高电平时有效
D1 [31:0]	I	32位输入信号，代表源操作数1
D2 [31:0]	I	32位输入信号，代表源操作数2
HILO_Op [3:0]	I	4位输入信号，选择不同的乘除法指令对应的操作
HILO_Result [31:0]	O	32位输出信号，代表HILO需要从HI寄存器或LO寄存器输出的数据
HILO_Busy	O	1位输出信号，表示HILO是否正在执行乘除运算操作

8.M_BE模块：

信号名称	方向	功能
A [1:0]	I	2位输入信号，代表地址的后两位
DM_Op [3:0]	I	4位输入信号，代表执行的存取操作
m_data_byteen [3:0]	O	4位输出信号，用于写入内存的字节使能

9.CP0模块:

信号名称	方向	功能
clk	I	时钟信号
reset	I	同步复位信号
A1[4:0]	I	5位输入信号，读CP0寄存器编号
A2[4:0]	I	5位输入信号，写CP0寄存器编号
CP0In[31:0]	I	32位输入信号，CP0寄存器的写入数据
PC[31:0]	I	32位输入信号，中断异常发生时的受害地址或受害地址+4
BDIn	I	1位输入信号，判断是否为延迟槽指令
ExcCode[4:0]	I	5位输入信号，中断异常的类型
HWInt[5:0]	I	6位输入信号，外设输入中断信号
We	I	1位输入信号，CP0写使能信号
EXLClr	I	1位输入信号，异常中断处理终止
Req	O	1位输出信号，中断异常请求
EPCOut[31:0]	O	32位输出信号，EPC寄存器的值
CP0Out[31:0]	O	32位输出信号，CP0的读出值

10.W_DataEXT模块：

信号名称	方向	功能
DM_Op_W [3:0]	I	4位输入信号， 代表执行的存取操作
Din [31:0]	I	32位输入信号，代表从DM中读出的数据
A [1:0]	I	2位输入信号，代表取址的地址后两位
Dout [31:0]	O	32位输出信号，代表数据拓展后的结果

11.D_REG寄存器

```
module D_REG(

input clk,

input reset,
```

```
input en,

input [31:0] Instr_F,

input [31:0] PC_F,



output reg [31:0] Instr_D,

output reg [31:0] PC_D,

output reg [25:0] Instr_index_D,

output reg [15:0] Imm_D,

output reg [4:0] A1_D,

output reg [4:0] A2_D

);
```

12.E_REG寄存器

```
module E_REG(

input clk,
input reset,
input clr,

//come from MainCU
input [3:0] ALUOp,
input ALUSrc_Sel,
input RegWrite,
input [1:0] Elevel_Sel,
input [1:0] Mlevel_Sel,
input [1:0] Wlevel_Sel,

 //come from D_stage
input [31:0] Instr_D,
input [31:0] PC_D,
input [31:0] EXTOut,
input [4:0] A1_D,
input [4:0] A2_D,

 //come from Decoder
```

```
input [4:0] A3,
input [2:0] Tnew,

 //come from forward-Mux
input [31:0] MF_Rs_D,
input [31:0] MF_Rt_D,

//control signal
output reg [3:0] ALUOp_E,
output reg [3:0] HILO_Op_E,
output reg [3:0] DM_Op_E,
output reg start_E,
output reg ALUSrc_Sel_E,
output reg RegWrite_E,
output reg [1:0] Elevel_Sel_E,   //E级选择数据转发
output reg [1:0] Mlevel_Sel_E,   //M级选择数据转发
output reg [1:0] Wlevel_Sel_E,   //W级选择数据转发

output reg [31:0] Instr_E,
output reg [31:0] PC_E,
output reg [31:0] V1_E,
output reg [31:0] V2_E,
output reg [31:0] E32_E,

output reg [4:0] A1_E,
output reg [4:0] A2_E,
output reg [4:0] A3_E,
output reg [2:0] Tnew_E
);
```

13.M_REG寄存器

```
module M_REG(
input clk,
input reset,


input RegWrite_E,
input [1:0] Mlevel_Sel_E,
input [1:0] Wlevel_Sel_E,

input [31:0] Instr_E,
input [31:0] PC_E,
input [31:0] ALUOut,
input [31:0] V2_E,
```

```
input [4:0] A1_E,
input [4:0] A2_E,
input [4:0] A3_E,
input [2:0] Tnew_E,
input [31:0] HILO_Result,
input [3:0] DM_Op_E,

output reg MemWrite_M,
output reg RegWrite_M,
output reg [1:0] Mlevel_Sel_M,  //M阶段转发数据选择信号
output reg [1:0] Wlevel_Sel_M,

output reg [31:0] Instr_M,
output reg [31:0] PC_M,
output reg [31:0] AO_M,     //写回寄存器的WriteData
output reg [31:0] V2_M,     //输入到DM中的WriteData

output reg [4:0] A1_M,
output reg [4:0] A2_M,
output reg [4:0] A3_M,
output reg [2:0] Tnew_M,
output reg [31:0] HL_M,
output reg [3:0] DM_Op_M
);
```

14.W_REG寄存器

```
module W_REG(
input clk,
input reset,

input RegWrite_M,
input [3:0] DM_Op_M,
input [1:0] Wlevel_Sel_M,

input [31:0] PC_M,
input [31:0] Instr_M,
input [31:0] AO_M,
input [31:0] DM_RD,

input [4:0] A1_M,
input [4:0] A2_M,
input [4:0] A3_M,
input [31:0] HL_M,
```

```
output reg RegWrite_W,
output reg [3:0] DM_Op_W,
output reg [31:0] Instr_W,
output reg [1:0] Wlevel_Sel_W,     //W阶段转发数据选择信号

output reg [31:0] PC_W,
output reg [31:0] AO_W,
output reg [31:0] DR_W,

output reg [4:0] A1_W,
output reg [4:0] A2_W,
output reg [4:0] A3_W,
output reg [31:0] HL_W
);
```

15.Bridge：

信号名称	方向	功能
cpu_m_data_addr[31:0]	I	32位输入信号，CPU输出的数据地址
cpu_m_data_wdata[31:0]	I	32位输入信号，CPU输出的写数据
cpu_m_data_byteen[3:0]	I	4位输入信号，CPU输出的字节写使能
cpu_m_data_rdata[31:0]	O	32位输出信号，往CPU中输入的数据
m_data_addr[31:0]	O	32位输入信号，mips微系统输出的数据地址
m_data_wdata[31:0]	O	32位输入信号，mips微系统输出的写数据
m_data_byteen[3:0]	O	4位输入信号，mips为系统发送输出的字节写使能
m_data_rdata[31:0]	I	32位输出信号，往mips微系统中输入的数据
m_int_addr[31:0]	O	32位输出信号，中断发生器待写入地址
m_int_byteen[3:0]	O	4位输出信号，中断发生器字节使能信号
DEV_Addr[31:0]	O	32位输出信号，写入计时器的地址
DEV_WD[31:0]	O	32位输出信号，写入计时器的数据
DEV0_RD[31:0]	I	32位输入信号，从DEV0读出的数据
DEV1_RD[31:0]	I	32位输入信号，从DEV1读出的数据
WeCPU	I	1位输入信号，计时器写使能信号
WeDEV0	O	1位输出信号，DEV0写使能信号
WeDEV1	O	1位输出信号，DEV1写使能信号

16.DEV：

信号名称	方向	功能
clk	I	时钟信号
reset	I	同步复位信号
Add[31:2]	I	30位输入信号，地址的高30位
WE	I	1位输入写使能信号
Din[31:0]	I	32位输入信号，向DEV中写入的数据
Dout[31:0]	O	32位输出信号，从DEV中读出的数据
IRQ	O	1位输出信号，代表外部中断信号

测试数据

异常处理程序：

```
.ktext 0x4180
_entry:
  mfc0  $1, $13
  ori   $k0, $0, 0x1000
  sw    $sp, -4($k0)

  addi  $k0, $k0, -256
  addi  $sp, $k0,0

  beq $0,$0,_save_context
  nop

_main_handler:
  mfc0  $k0, $13              #读出异常编码
    ori $k1, $0, 0x7c
    and $k0, $k0, $k1

  ori $k1, $0, 0x0000         #中断
  beq $k0, $k1, int_handler
  nop

  ori $k1, $0, 0x0004         #取指令/数据错误
  beq $k0, $k1, adel_handler
  nop
  ori $k1, $0, 0x0005         #存数据错误
  beq $k0, $k1, ades_handler
  nop
  ori $k1, $0, 0x000a         #未知指令
  beq $k0, $k1, ri_handler
  nop
  ori $k1, $0, 0x000c         #溢出
  beq $k0, $k1, ov_handler
  nop

int_handler:
  beq $0,$0,_restore_context  #直接返回
  nop

adel_handler:
  mfc0  $t0, $14
  mfc0  $k0, $13
  lui $t2, 0x8000   #is delay_slot?
  and $t3, $k0, $t2
```

```
  addi  $t0, $t0, 4
  bne $t3, $t2, adel_nxt
  nop
  addi  $t0, $t0, 4
  adel_nxt:
  mtc0  $t0, $14
  beq $0,$0,_restore_context
  nop

ades_handler:
  mfc0  $t0, $14
  mfc0  $k0, $13
  lui $t2, 0x8000
  and $t3, $k0, $t2
  addi  $t0, $t0, 4
  bne $t3, $t2, ades_nxt
  nop
  addi  $t0, $t0, 4
  ades_nxt:
  mtc0  $t0, $14
  beq  $0,$0,_restore_context
  nop


ri_handler:
  mfc0  $t0, $14
  mfc0  $k0, $13
  lui $t2, 0x8000
  and $t3, $k0, $t2
  addi  $t0, $t0, 4
  bne $t3, $t2, ri_nxt
  nop
  addi  $t0, $t0, 4
  ri_nxt:
  mtc0  $t0, $14
  beq $0,$0,_restore_context
  nop

ov_handler:
  mfc0  $t0, $14
  mfc0  $k0, $13
  lui $t2, 0x8000
  and $t3, $k0, $t2
  addi  $t0, $t0, 4
  bne $t3, $t2, ov_nxt
  nop
  addi  $t0, $t0, 4
  ov_nxt:
  mtc0  $t0, $14
```

```
  beq $0,$0,_restore_context
  nop

_restore:
  eret

_save_context:
      sw    $2, 8($sp)
      sw    $3, 12($sp)
      sw    $4, 16($sp)
      sw    $5, 20($sp)
      sw    $6, 24($sp)
      sw    $7, 28($sp)
      sw    $8, 32($sp)
      sw    $9, 36($sp)
      sw    $10, 40($sp)
      sw    $11, 44($sp)
      sw    $12, 48($sp)
      sw    $13, 52($sp)
      sw    $14, 56($sp)
      sw    $15, 60($sp)
      sw    $16, 64($sp)
      sw    $17, 68($sp)
      sw    $18, 72($sp)
      sw    $19, 76($sp)
      sw    $20, 80($sp)
      sw    $21, 84($sp)
      sw    $22, 88($sp)
      sw    $23, 92($sp)
      sw    $24, 96($sp)
      sw    $25, 100($sp)
      sw    $28, 112($sp)
      sw    $29, 116($sp)
      sw    $30, 120($sp)
      sw    $31, 124($sp)
  mfhi  $k0
  sw  $k0, 128($sp)
  mflo  $k0
  sw  $k0, 132($sp)
  beq $0,$0,_main_handler
  nop


_restore_context:
  addi  $sp, $0,  0x1000
  addi  $sp, $sp, -256
      lw    $2, 8($sp)
      lw    $3, 12($sp)
      lw    $4, 16($sp)
```

```
      lw    $5, 20($sp)
      lw    $6, 24($sp)
      lw    $7, 28($sp)
      lw    $8, 32($sp)
      lw    $9, 36($sp)
      lw    $10, 40($sp)
      lw    $11, 44($sp)
      lw    $12, 48($sp)
      lw    $13, 52($sp)
      lw    $14, 56($sp)
      lw    $15, 60($sp)
      lw    $16, 64($sp)
      lw    $17, 68($sp)
      lw    $18, 72($sp)
      lw    $19, 76($sp)
      lw    $20, 80($sp)
      lw    $21, 84($sp)
      lw    $22, 88($sp)
      lw    $23, 92($sp)
      lw    $24, 96($sp)
      lw    $25, 100($sp)
      lw    $28, 112($sp)
      lw    $30, 120($sp)
      lw    $31, 124($sp)
  lw    $k0, 128($sp)
  mthi  $k0
  lw    $k0, 132($sp)
  mtlo  $k0
    lw    $29, 116($sp)
  ori     $1,$0,1
    beq $0,$0,_restore
  nop
```

异常测试主程序：

```
.text
    # 只允许外部中断
    ori $t0, $0, 0x1001
    mtc0 $t0, $12

    #ov
    lui $t0, 0x7fff
    lui $t1, 0x7fff

    add $t2, $t0, $t1
    addi $t2,$t0,0x7fff
    lui $t0,0xffff
    lui $t1,0xffff
```

```
sub $t2,$t0,$t1

#adel
  #ALign
lb $0,-1($0)
lb $0,-2($0)
lb $0,-3($0)
lb $0,0($0)
lb $0,1($0)
lb $0,2($0)
lb $0,3($0)

lh $0,-1($0)
lh $0,-2($0)
lh $0,-3($0)
lh $0,0($0)
lh $0,1($0)
lh $0,2($0)
lh $0,3($0)

lw $0,-1($0)
lw $0,-2($0)
lw $0,-3($0)
lw $0,0($0)
lw $0,1($0)
lw $0,2($0)
lw $0,3($0)

  #Timer
ori $t0,$0,0x7f00
lb $0,0($t0)
lh $0,0($t0)

  #Range
ori $t0,$0,0x3000
lb $0,0($t0)
lh $0,0($t0)
lw $0,0($0)

#ades
sb $0,-1($0)
sb $0,-2($0)
sb $0,-3($0)
sb $0,0($0)
sb $0,1($0)
sb $0,2($0)
sb $0,3($0)

sh $0,-1($0)
```

```
    sh $0,-2($0)
    sh $0,-3($0)
    sh $0,0($0)
    sh $0,1($0)
    sh $0,2($0)
    sh $0,3($0)

    sw $0,-1($0)
    sw $0,-2($0)
    sw $0,-3($0)
    sw $0,0($0)
    sw $0,1($0)
    sw $0,2($0)
    sw $0,3($0)

      #Timer
    ori $t0,$0,0x7f00
    sb $0,0($t0)
    sh $0,0($t0)

      #Range
    ori $t0,$0,0x3000
    sb $0,0($t0)
    sh $0,0($t0)
    sw $0,0($0)

      #Count
    ori $t0,$0,0x7f08
    sb $0,0($0)
    sh $0,0($0)
    sw $0,0($0)


    #syscall
    ori $t0,0
    syscall
    ori $t1,0


    #delay_slot_test
    lui $t0, 0x7fff
    lui $t1, 0x7fff
    jal label:
    add $t2, $t0, $t1

label:
  ori $t0,0
  ori $t1,0
```

思考题

1、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?

鼠标和键盘此类的低速设备是通过中断请求的方式进行操作的：当键盘按下一个按键或者鼠标点击的时候，此类外部设备会发出一个中断信号(interrupt)，中断信号经过中断控制器传到CPU，然后CPU根据不同的中断信号执行不同的中断响应程序。

2、请思考为什么我们的 CPU 处理中断异常必须是已经指定好的地址？如果你的 CPU 支持用户自定义入口地址，即处理中断异常的程序由用户提供，其还能提供我们所希望的功能吗？如果可以，请说明这样可能会出现什么问题？否则举例说明。（假设用户提供的中断处理程序合法）

因为如果不提前规定好处理中断异常程序的地址入口，当代码量过大的时候，会存在代码指令与异常处理程序指令地址相冲突的情况。如果CPU支持用户自定义入口地址，那么在不产生主程序与异常处理程序地址冲突的前提下，能够提供我们所希望的功能。但是如果产生地址冲突，就会产生程序出错。

3、为何与外设通信需要 Bridge?

因为在实际设计中，CPU需要与多个外设进行通信，为每一个外设在CPU处增加端口显然是不合实际的。故在CPU与外设之间增加Bridge作为“物流分散站”的作用，CPU将数据交给Bridge，由Bridge进行数据分发至不同外设。当外设产生中断时，也可以先到Bridge处进行整合再输入至CPU中。

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移图。

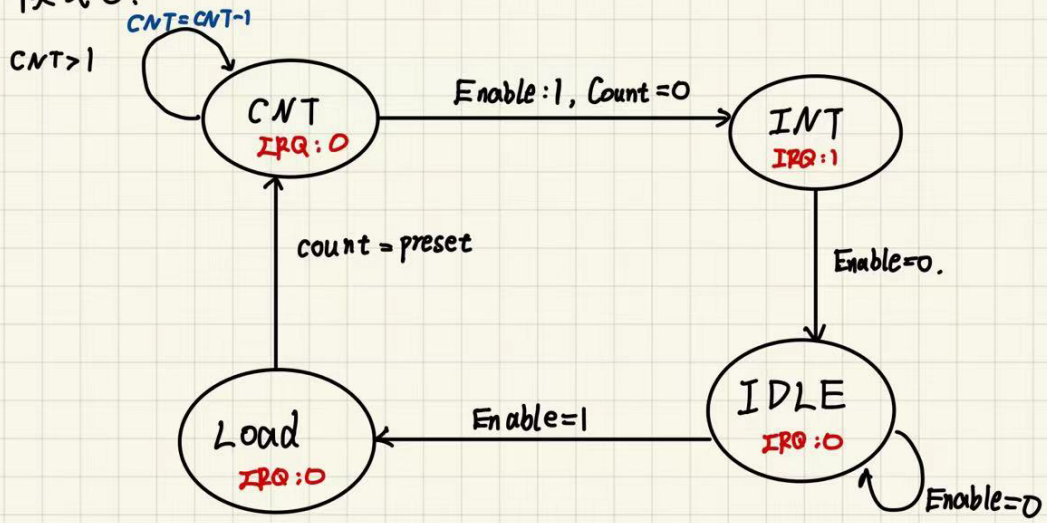
两种中断模式的异同：

模式0：当使能Enable被设置为1时，计数器将初值寄存器的值写入计数器并开始倒计时。计数器倒计数为 0 后，计数器停止计数，同时Enable值变为0。模式0常用于产生定时中断

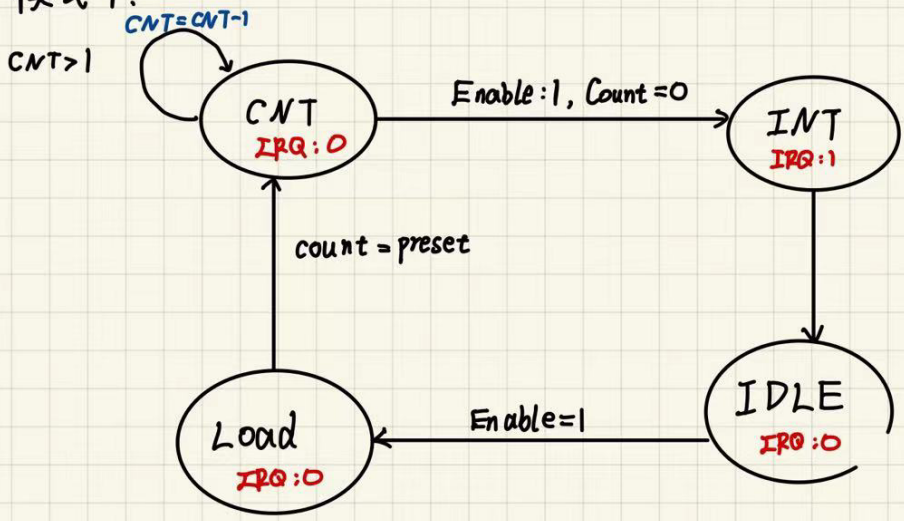
模式1：当Enable呗设置为1时，计数器持续倒计时。当计数器倒计数为 0 后，初值寄存器值被写入至计数器，从而周期性地产生中断。

状态转移图：

模式0：


CNT=CNT-1
CNT>1
CNT
IRQ:0
Enable:1, Count=0
INT
IRQ:1
count = preset
Enable=0.
IDLE
IRQ:0
Load
IRQ:0
Enable=1
Enable=0

模式1：


CNT=CNT-1
CNT>1
CNT
IRQ:0
Enable:1, Count=0
INT
IRQ:1
count = preset
IDLE
IRQ:0
Load
IRQ:0
Enable=1

5、倘若中断信号流入的时候，在检测宏观 PC 的一级如果是一条空泡（你的 CPU 该级所有信息均为空）指令，此时会发生什么问题？在此例基础上请思考：在 P7 中，清空流水线产生的空泡指令应该保留原指令的哪些信息？

此时宏观PC的地址会为0x0000_0000，如果此时中断信号流入，往EPC中写入的受害者地址将是0x0000_000，那么从异常处理程序返回的时候将返回到错误的地址，导致程序出错。

应保留的信息：指令的地址PC，是否为延迟槽指令BD，异常编码ExcCode。

6、为什么 `jalr` 指令为什么不能写成 `jalr $31, $31`？

因为该指令在重复执行的时候可能会产生不同的结果，jalr的延迟槽的指令发生异常时，在处理完异常后将无法正常返回到主程序出错的地址。

7、[P7 选做] 请详细描述你的测试方案及测试数据构造策略。

测试方案：针对教程中不同的异常情况进行测试，结合输出数据与波形图判断是否正确运行异常处理程序。

数据构造：针对不同类型的错误编号，构造数据溢出、地址错误（具体为教程中的异常类型）进行异常构造。