

计算机组成原理实验报告

一、CPU 设计方案综述

(一) 总体设计概述

本 CPU 为 verilog 实现的五级流水线 CPU，支持的指令集包括 {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、mfc0、mtc0、eret}，CPU 五级分别为 F 级，D 级，E 级，M 级，W 级。为了实现这些功能，CPU 主要包含了 pc，im，grf，ext，comp，alu，mult_div，CP0，bridge，dm，mux，ctrl，Hazard。

指令分类：

寄存器立即数计算：addi, addiu, slti, sltiu, andi, ori, xori, sll, srl, sra

双寄存器计算：add, addu, sub, subu, slt, sltu, and, or, nor, xor, sllv, srlv, srav

根据寄存器分支：beq, bne, bgez, bgtz, blez, bltz

写内存：sw, sh, sb

读内存：lw, lh, lhu, lb, lbu

读乘除法寄存器：mfhi, mflo

写乘除法寄存器：mthi, mtlo, mult, multu, div, divu

跳转并链接：jal, jalr

跳转寄存器：jr, jalr

加载高位：lui

空指令：nop

(二) 关键模块定义

表 1 pc

信号名	方向	描述
stall	I	暂停信号（PC 值保持不变）
D_PC[31:0]	I	D 级 PC
ext_data[31:0]	I	提供计算 beq 地址的数据
imm26[25:0]	I	提供计算 jal,j 地址的数据
reg_data[31:0]	I	jr 指令地址
next_pc	中间变量	下一条指令地址
NPCSrc[2:0]	I	指令地址选择信号
clk	I	时钟信号
reset	I	同步复位信号
pc	O	当前指令地址

NPCSrc（控制信号）	功能描述
000	pc+4
001	选择 beq 指令地址
010	选择 jal 指令地址
011	选择 jr 指令地址

表 2 im

信号名	方向	描述
pc[31:0]	I	指令地址
instr[31:0]	O	指令信号

表 3 grf

信号名	方向	描述
PC	I	W 级 PC
read_reg1[4:0]	I	读出寄存器 1 地址
read_reg2[4:0]	I	读出寄存器 2 地址
write_reg[4:0]	I	写入地址
wdata[31:0]	I	写入数据

clk	I	时钟信号
reset	I	同步复位信号
RegWrite	I	写入使能端信号
rdata1[31:0]	O	读出寄存器 1 数据
rdata2[31:0]	O	读出寄存器 2 数据

表 4 ext

信号名	方向	描述
omml6[15:0]	I	16 位立即数
ExtOp[2:0]	I	扩展选择信号
ext_Data[31:0]	O	扩展结果

ExtOp (控制信号)	功能描述
000	16 位立即数零扩展至 32 位
001	16 位立即数符号扩展至 32 位
011	16 位立即数加载至高位

表 5 comp

信号名	方向	描述
A[31:0]	I	第一个比较数
B[31:0]	I	第二个比较数
CompOp[3:0]	I	比较选择信号
out	O	比较结果

CompOp (控制信号)	功能描述
0000	等于
0001	不等于
0010	大于
0011	小于
0100	大于等于
0101	小于等于

表 6 alu

信号名	方向	描述
A[31:0]	I	第一个运算数
B[31:0]	I	第二个运算数
shamt[4:0]	I	移位数
ALUOp[3:0]	I	运算选择信号
C[31:0]	O	运算结果

ALUOp (控制信号)	功能描述
0000	加
0001	减
0010	按位与
0011	按位或
0100	按位异或
0101	逻辑左移
0110	逻辑右移
0111	算术右移

表 7 mult_div

信号名	方向	描述
A[31:0]	I	第一个运算数
B[31:0]	I	第二个运算数
clk	I	时钟信号
reset	I	同步复位信号
start	I	start 信号
MultDivOp[3:0]	I	运算选择信号
MDUWrite	I	写使能信号
req	I	发生中断或异常信号
busy	O	busy 信号
hi[31:0]	O	HI 寄存器数据
lo[31:0]	O	LO 寄存器数据

表 8 CP0

信号名	方向	描述
Addr[4:0]	I	CP0 读写寄存器的地址
DIn[31:0]	I	CP0 写数据
PC[31:0]	I	受害指令的 PC

excCode[6:2]	I	异常信号
HWInt[5:0]	I	中断信号
bd	I	BD 信号
WE	I	CP0 写使能信号
EXLClr	I	将 EXL 置 0
clk	I	时钟信号
rst	I	同步复位信号
IntReq	O	响应中断信号
ExcReq	O	响应异常信号
Req	O	IntReq ExcReq
t_Req	O	响应 interrupt 中断信号
epc[31:0]	O	EPC 地址
DOut[31:0]	O	CP0 读数据

寄存器	功能域	描述
SR	IM[15:10]	对应六个外部中断, 相应位置 1 表示允许中断, 置 0 表示禁止中断
	EXL[1]	任何异常发生时置位
	IE[0]	全局中断使能位
Cause	BD[31]	若发生异常的指令在延迟槽, 则置位
	IP[15:10]	对应六个外部中断, 相应位置 1 表示有中断, 置 0 表示无中断
	ExcCode[6:2]	异常编码
EPC		保存异常返回点的 PC
PRId		用于实现个性的编码

表 9 bridge

信号名	方向	描述
CPU_Addr[31:0]	I	CPU 需要读写的地址
CPU_byteen[3:0]	I	CPU 写使能信号
CPU_WD[31:0]	I	CPU 需要写的的数据
DM_RD[31:0]	I	DM 读出的数据
TIMER0_RD[31:0]	I	TC0 读出的数据
TIMER1_RD[31:0]	I	TC1 读出的数据
DEV_Addr[31:0]	O	TC 和 DM 需要读写的地址
DM_byteen[3:0]	O	DM 写使能信号
DM_WD[31:0]	O	DM 需要写的的数据
TIMER0_WE	O	TC0 写使能信号
TIMER1_WE	O	TC1 写使能信号

DEV_RD[31:0]	O	外设最终读出的数据（给回 CPU）
--------------	---	-------------------

表 10 dm

信号名	方向	描述
PC	I	M 级 PC
addr[31:0]	I	写入或读出地址
wdata[31:0]	I	写入数据
MemWrite	I	写使能端
clk	I	时钟信号
reset	I	同步复位信号
rdata[31:0]	O	读出数据

表 11 mux

信号名	方向	描述
rt[4:0]	I	GRF 写入地址端选择 1 （W 级）
rd[4:0]	I	GRF 写入地址端选择 2 （W 级）
alu[31:0]	I	GRF 写入数据端选择 1 （W 级）
dm[31:0]	I	GRF 写入数据端选择 2 （W 级）
ext[31:0]	I	GRF 写入数据端选择 3 / （W 级） ALU 第二个运算数 B 端选择 2 （E 级）
PC+8[31:0]	I	GRF 写入数据端选择 4 （W 级）
grf_rdata2[31:0]	I	ALU 第二个运算数 B 端选择 1 （E 级）
RegDst[2:0]	I	GRF 写入地址端选择信号 （W 级）
MemtoReg[2:0]	I	GRF 写入数据端选择信号 （W 级）
ALUSrc[2:0]	I	ALU 第二个运算数 B 端选择信号 （E 级）
grf_wreg[4:0]	O	GRF 写入地址端最终选择 （W 级）
grf_wdata[31:0]	O	GRF 写入数据端最终选择 （W 级）
alu_b[31:0]	O	ALU 第二个运算数 B 端最终选择 （E 级）

注：括号里为该信号名由该级使用

表 12 ctrl

信号名	方向	描述
opcode[5:0]	I	Opcode 信号
func[5:0]	I	Func 信号

comp	I	比较结果信号
RegDst[2:0]	O	GRF 写入地址端选择信号 (W 级)
RegWrite	O	GRF 写入使能端 (W 级)
MemWrite	O	DM 写入使能端 (M 级)
MemtoReg[2:0]	O	GRF 写入数据端选择信号 (W 级)
ALUOp[3:0]	O	ALU 运算选择信号 (E 级)
ALUSrc[2:0]	O	ALU 第二个运算数 B 端选择信号 (E 级)
NPCSrc[2:0]	O	PC 输入端选择信号 (D 级)
CompOp[3:0]	O	Comp 比较选择信号 (D 级)
ExtOp[2:0]	O	Ext 输入端选择信号 (D 级)
Tuse_rs[2:0]	O	Tuse_rs 信号 (D 级)
Tuse_rt[2:0]	O	Tuse_rt 信号 (D 级)
Tnew[2:0]	O	Tnew 信号 (E 级)

注：括号里为该信号名由该级使用

表 13 Hazard

信号名	方向	描述
D_Tuse_rs[2:0]	I	D 级 Tuse_rs 信号
D_Tuse_rt[2:0]	I	D 级 Tuse_rt 信号
E_Tnew[2:0]	I	E 级 Tnew 信号
M_Tnew[2:0]	I	M 级 Tnew 信号
W_Tnew[2:0]	I	W 级 Tnew 信号
D_ir[31:0]	I	D 级 instr 信号
E_ir[31:0]	I	E 级 instr 信号
M_ir[31:0]	I	M 级 instr 信号
W_ir[31:0]	I	W 级 instr 信号
E_ext[31:0]	I	E 级 ext 数据
E_PC8[31:0]	I	E 级 PC+8 数据
M_alu[31:0]	I	M 级 alu 数据
M_ext[31:0]	I	M 级 ext 数据
M_PC8[31:0]	I	M 级 PC+8 数据
W_alu[31:0]	I	W 级 alu 数据
W_dm[31:0]	I	W 级 dm 数据
W_ext[31:0]	I	W 级 ext 数据
W_PC8[31:0]	I	W 级 PC+8 数据
D_grf_rdata1[31:0]	I	D 级 grf_rdata1 数据
D_grf_rdata2[31:0]	I	D 级 grf_rdata2 数据
E_grf_rdata1[31:0]	I	E 级 grf_rdata1 数据
E_grf_rdata2[31:0]	I	E 级 grf_rdata2 数据
M_grf_rdata2[31:0]	I	M 级 grf_rdata2 数据

MFRSD[31:0]	O	该数据转发至 D 级 rs 端
MFRTD[31:0]	O	该数据转发至 D 级 rt 端
MFRSE[31:0]	O	该数据转发至 E 级 rs 端
MFRTE[31:0]	O	该数据转发至 E 级 rt 端
MFRTM[31:0]	O	该数据转发至 M 级 rt 端
stall	O	暂停信号

表 14 执行该指令所需要的控制信号

	执行指令									
Opcode	000000	000000	001111	001101	101011	100011	000100	000011	000000	000000
func	100001	100011							001000	000000
信号名	Addu	subu	lui	ori	sw	lw	beq	jal	jr	nop
RegDst	001	001	000	000				010		
RegWrite	1	1	1	1		1		1		
MemWrite					1					
MemtoReg	000	000	010	000		001		011		
ALUOp	0000	0001		0011	0000	0000				
ALUSrc	000	000		001	001	001				
NPCSrc	000	000	000	000	000	000	001	010	011	
CompOp							0000			
ExtOp			011	000	001	001	001	010		

表 15 标记 x 代表对应流水级需要设置相应寄存器

信号名	功能	D 级 (IF/ID)	E 级 (ID/EX)	M 级 (EX/MEM)	W 级 (MEM/WB)
instr	传递指令	x	x	x	x
PC8	下一条指令地址(jal/jalr)	x	x	x	x
grf_rdata1	grf 的 rs 值(cal_r)		x		
grf_rdata2	grf 的 rt 值(store)		x	x	
ext	扩展结果(lui)		x	x	x
alu	计算结果(cal_r)			x	x
dm_rdata	dm 读出数据(load)				x
Tnew	Tnew 信号			x	x

(三) 重要机制实现方法

1.跳转

图 1 ctrl 模块中的某部分

```
assign NPCSrc = (branch & comp) ? 3'b01 :
                (jal | j) ? 3'b10 :
                (jr) ? 3'b11 : 3'b00;
```

comp 模块和 ctrl 模块协同工作产生 NPCSrc 信号。

图 2 pc 模块中的某部分

```
wire [31:0] D_pc = ((D_PC - 32'h3000) >> 2);
wire [31:0] j_target = {D_PC[31:28],imm26,2'b0};

wire [31:0] b_addr = D_pc + 1 + ext_data;
wire [31:0] j1_addr = ((j_target - 32'h3000) >> 2); //jal,j
wire [31:0] j2_addr = ((reg_data - 32'h3000) >> 2); //jr
wire [31:0] next_pc;

assign next_pc = (NPCSrc == 3'b01) ? b_addr :
                 (NPCSrc == 3'b10) ? j1_addr :
                 (NPCSrc == 3'b11) ? j2_addr : pc + 1;

always@(posedge clk) begin
    if(reset) pc <= 32'b0;
    else if(stall) pc <= pc; //pc.en = 0
    else pc <= next_pc;
end
```

pc 模块内置了计算单元,并配合 NPCSrc 信号来支持指令 beq,jal,j,jr 的跳转机制。

2.转发

一条指令的 Tuse 信号在 D 级生成。
一条指令的 Tnew 信号在 E 级生成并流水,每经过一级其 Tnew 值减一,直到等于零为止。

表 16 Tuse 表

IF/ID 当前指令		
指令类型	源寄存器	Tuse

branch	rs/rt	0
jump	rs	0
Cal_r	rs/rt	1
Cal_i	rs	1
load	rs	1
store	rs	1
store	rt	2

*jump 指 jalr 和 jr

表 17 Tnew 表(Tnew 值/寄存器)

Tnew											
ID/EX				EX/MEM				MEM/WB			
Cal_r	Cal_i	Load	Jal	Cal_r	Cal_i	Load	Jal	Cal_r	Cal_i	Load	Jal
1/rd	1/rt	2/rt	0/\$31	0/rd	0/rt	1/rt	0/\$31	0/rd	0/rt	0/rt	0/\$31
	Lui		Jalr		Lui		Jalr		Lui		Jalr
	0/rt		0/rd		0/rt		0/rd		0/rt		0/rd

若 Tnew = 0，表示该数据已生成，可转发。
MFRSD 转发至 D 级 comp 模块的 A 输入端。
MFRTD 转发至 D 级 comp 模块的 B 输入端。
MFRSE 转发至 E 级 alu 模块的 A 输入端。
MFRTE 转发至 E 级 alu 模块的 B 输入端之一。
MFRTM 转发至 M 级 dm 模块的写入数据端。

3.暂停

IF/ID 当前指令			ID/EX (Tnew)			EX/MEM (Tnew)
指令类型	源寄存器	Tuse	Cal_r 1/rd	Cal_i 1/rt	Load 2/rt	Load 1/rt
branch	rs/rt	0	暂停	暂停	暂停	暂停
jump	rs	0	暂停	暂停	暂停	暂停
Cal_r	rs/rt	1			暂停	
Cal_i	rs	1			暂停	
load	rs	1			暂停	
store	rs	1			暂停	

若 Tnew > Tuse，则暂停。同时产生 stall 信号，冻结 IF/ID(D 级流水线寄存器)，清除 ID/EX(E 级流水线寄存器)，禁止 PC。
(在 D 级的时候会通过 Tuse 和 Tnew 值判断是否需要暂停)

图 3 CPU (参考)

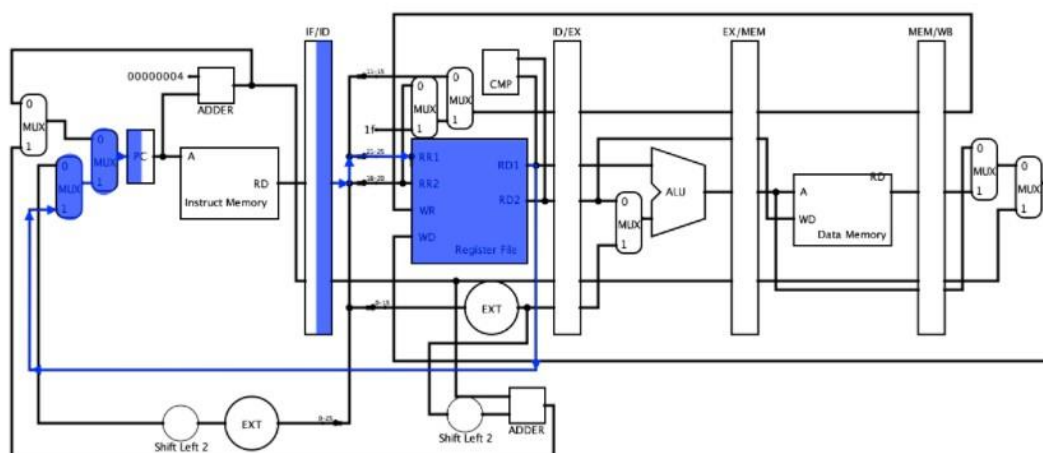
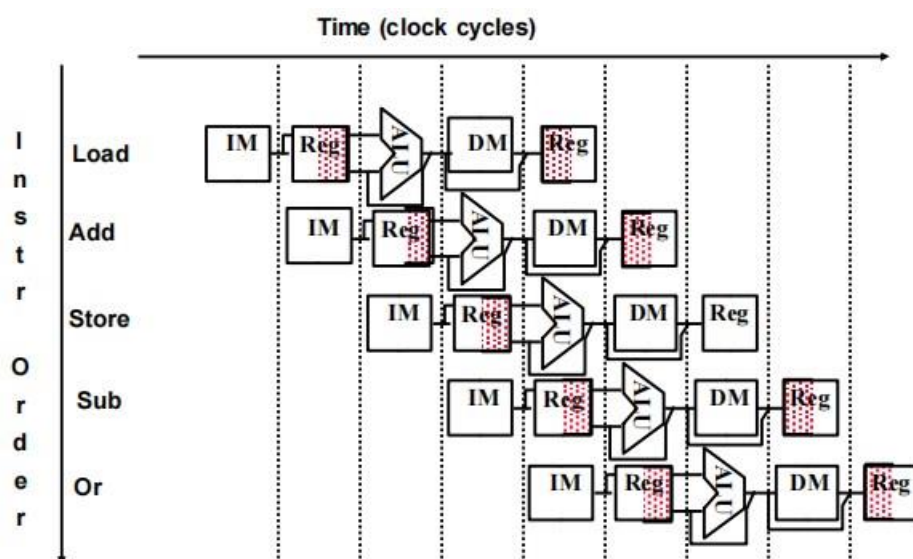


图 4 五条指令在时钟周期上的执行过程 (参考)










二、测试方案

(一) 典型测试样例

使用了讨论区分享的 P7 测试数据。

eret测试	2021/12/19 10:04	File folder
interrupt测试	2021/12/18 23:34	File folder
非异常边界测试	2021/12/18 23:20	File folder
异常测试	2021/12/18 23:25	File folder

 P7exception	2021/12/21 20:02	File folder	
 P7interrupt	2021/12/21 18:42	File folder	
 check.py	2021/12/2 19:38	JetBrains PyCharm ...	2 KB
 dump.py	2021/12/21 19:24	JetBrains PyCharm ...	3 KB
 ktext.txt	2021/12/21 20:03	Text Document	1 KB
 readme.txt	2021/12/21 18:31	Text Document	1 KB
 text.txt	2021/12/21 20:03	Text Document	1 KB

1. 异常测试

汇编代码:

```
.text
ori $t0,$zero,0x1001

mtc0 $t0,$12    #允许 interrupt 中断

ori $t1,$zero,0x4532

sw $t1,2($0)    #AdES_5 存数地址未与 4 字节对齐

lw $t2,3($0)    #AdEL_4 取数地址未与 4 字节对齐

ori $t1,$zero,0x8888

sh $t1,1($0)    #AdES_5 存数地址未与 2 字节对齐

lhu $t2,1($0)   #AdEL_4 取数地址未与 2 字节对齐

lh $t3,0($0)

lui $t0,0xffff
ori $t1,$t0,0xffff
ori $t5,$zero,0x303c
jr $t5

add $t3,$t1,$t1  #Ov_12 算术溢出

nop
ori $t2,$zero,0x3232

sw $t2,880($t1)  #AdES_5 计算地址加法溢出

lw $t2,880($t1)  #AdEL_4 计算地址加法溢出

madd $t2,$t2     #RI_10 未知的指令码

ori $t0,$zero,0x305d
```

```

jalr $ra,$t0          #AdEL_4 PC 地址未字对齐

nop
nop
nop                   #0x305c AdEL_4
ori $t0,$zero,0x7000

jr $t0                #AdEL_4 PC 地址超过 0x3000~0x6ffc

nop
add $t0,$zero,$t0
end:
beq $0,$0,end
nop
.ktext 0x4180
mfc0 $k0,$12
mfc0 $k0,$13
mfc0 $k0,$14
ori $k1,$zero,0x6ffc
bgt $k0,$k1,jump
nop
lui $k1,0xffff
ori $k1,$k1,0xfffc
and $k0,$k0,$k1
addi $k0,$k0,4
mtc0 $k0,$14
eret

jump:
ori $k1,$zero,0x3070
mtc0 $k1,$14
eret

```

机器码:

```

.text
34081001
40886000
34094532
ac090002
8c0a0003
34098888
a4090001
940a0001
840b0000
3c08ffff

```

3509ffff
340d303c
01a00008
01295820
00000000
340a3232
ad2a0370
8d2a0370
714a0000
3408305d
0100f809
00000000
00000000
00000000
34087000
01000008
00000000
00084020
1000ffff
00000000
.ktext
401a6000
401a6800
401a7000
341b6ffc
037a082a
14200007
00000000
3c1bffff
377bffc
035bd024
235a0004
409a7000
42000018
341b3070
409b7000
42000018

期望结果:

@00003000: \$ 8 <= 00001001
@00003008: \$ 9 <= 00004532
@00004180: \$26 <= 00001003
@00004184: \$26 <= 00000014
@00004188: \$26 <= 0000300c

@0000418c: \$27 <= 00006ffc
@00004190: \$ 1 <= 00000000
@0000419c: \$27 <= ffff0000
@000041a0: \$27 <= ffffffff
@000041a4: \$26 <= 0000300c
@000041a8: \$26 <= 00003010
@00004180: \$26 <= 00001003
@00004184: \$26 <= 00000010
@00004188: \$26 <= 00003010
@0000418c: \$27 <= 00006ffc
@00004190: \$ 1 <= 00000000
@0000419c: \$27 <= ffff0000
@000041a0: \$27 <= ffffffff
@000041a4: \$26 <= 00003010
@000041a8: \$26 <= 00003014
@00003014: \$ 9 <= 00008888
@00004180: \$26 <= 00001003
@00004184: \$26 <= 00000014
@00004188: \$26 <= 00003018
@0000418c: \$27 <= 00006ffc
@00004190: \$ 1 <= 00000000
@0000419c: \$27 <= ffff0000
@000041a0: \$27 <= ffffffff
@000041a4: \$26 <= 00003018
@000041a8: \$26 <= 0000301c
@00004180: \$26 <= 00001003
@00004184: \$26 <= 00000010
@00004188: \$26 <= 0000301c
@0000418c: \$27 <= 00006ffc
@00004190: \$ 1 <= 00000000
@0000419c: \$27 <= ffff0000
@000041a0: \$27 <= ffffffff
@000041a4: \$26 <= 0000301c
@000041a8: \$26 <= 00003020
@00003020: \$11 <= 00000000
@00003024: \$ 8 <= ffff0000
@00003028: \$ 9 <= ffffffff
@0000302c: \$13 <= 0000303c
@00003034: \$11 <= ffffffff
@0000303c: \$10 <= 00003232
@00004180: \$26 <= 00001003
@00004184: \$26 <= 00000014
@00004188: \$26 <= 00003040
@0000418c: \$27 <= 00006ffc

@00004190: \$ 1 <= 00000000
@0000419c: \$27 <= ffff0000
@000041a0: \$27 <= ffffffff
@000041a4: \$26 <= 00003040
@000041a8: \$26 <= 00003044
@00004180: \$26 <= 00001003
@00004184: \$26 <= 00000010
@00004188: \$26 <= 00003044
@0000418c: \$27 <= 00006ffc
@00004190: \$ 1 <= 00000000
@0000419c: \$27 <= ffff0000
@000041a0: \$27 <= ffffffff
@000041a4: \$26 <= 00003044
@000041a8: \$26 <= 00003048
@00004180: \$26 <= 00001003
@00004184: \$26 <= 00000028
@00004188: \$26 <= 00003048
@0000418c: \$27 <= 00006ffc
@00004190: \$ 1 <= 00000000
@0000419c: \$27 <= ffff0000
@000041a0: \$27 <= ffffffff
@000041a4: \$26 <= 00003048
@000041a8: \$26 <= 0000304c
@0000304c: \$ 8 <= 0000305d
@00003050: \$31 <= 00003058
@00004180: \$26 <= 00001003
@00004184: \$26 <= 00000010
@00004188: \$26 <= 0000305d
@0000418c: \$27 <= 00006ffc
@00004190: \$ 1 <= 00000000
@0000419c: \$27 <= ffff0000
@000041a0: \$27 <= ffffffff
@000041a4: \$26 <= 0000305c
@000041a8: \$26 <= 00003060
@00003060: \$ 8 <= 00007000
@00004180: \$26 <= 00001003
@00004184: \$26 <= 00000010
@00004188: \$26 <= 00007000
@0000418c: \$27 <= 00006ffc
@00004190: \$ 1 <= 00000001
@000041b4: \$27 <= 00003070

三、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？)

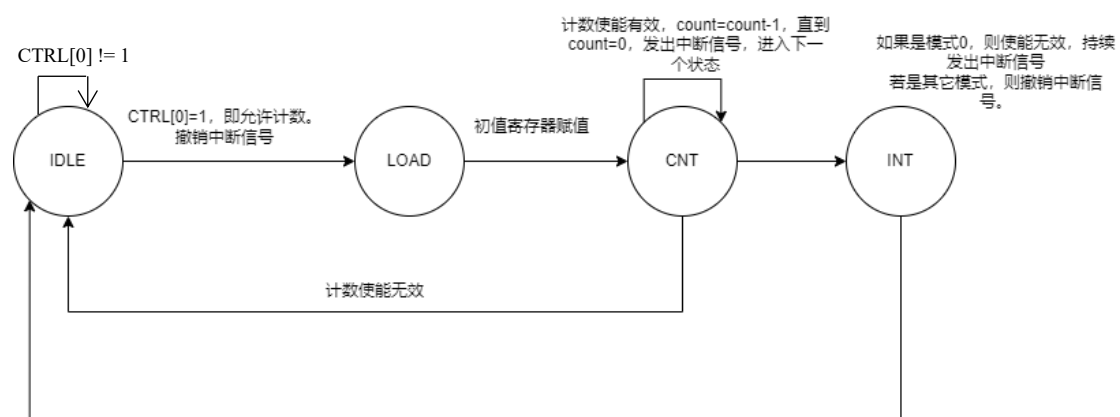
答：电脑等信息机器硬件组件间的接口叫硬件接口。电脑等信息机器软件组件间的接口叫软件接口。Bridge 模块就是接口，使 CPU 可以和外设交换数据。

2、BE 部件对所有的外设都是必要的吗？

答：不是必要的，BE 只对半字或字节的外设有用。

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：



相同：都能产生中断。

不同：模式 0 的中断信号能产生若干个周期，模式 1 的中断信号只产生一周期。

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) 定时器倒计时至 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。)

答：

```
.text
ori $t0,$zero,0x0c01

mtc0 $t0,$12      #将 SR 寄存器的 IM[11:10] 以及 IE 置 1

ori $s0,$zero,0x7f00
ori $t0,$zero,0x9
```

```

sw $t0,0($s0)      #将 TCO 的 CTRL 寄存器的 IM 以及 Enable 置 1

ori $s0,$zero,0x7f04
ori $t1,$zero,0x64

sw $t1,0($s0)      #将 TCO 的 PRESET 寄存器赋初值为 100

end:
beq $0,$0,end
nop

.ktext 0x4180
ori $s0,$zero,0x7f00
ori $t0,$zero,0x9

sw $t0,0($s0)      #再次启动定时器

eret

```

5、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：控制鼠标的过程，是在和操作系统交互，然后操作系统告诉 CPU 该计算什么。关于鼠标和操作系统的交互，在开机之后鼠标就有一个坐标，鼠标知道了自己向上移动了一英寸，只需要告诉系统:我向上移动了 2500 个点。然后操作系统得到新的坐标反馈到图形界面也就是显卡(显示器)，让这个界面反馈给我们。电脑需要时刻询问鼠标是否移动。（知乎）