

# P7 课下 & 课上总结

2021-12-15

首页 > 北航计算机组成原理

## H<sub>1</sub> P7: MIPS 微系统设计 #

通过阅读本文，您可以大致了解 2021 年秋季北航计算机组成课程 P7 课下搭建的思路，以及课上测试的题目内容、难度和解题思路

P7 课上测试的主要内容是对课下用 Verilog 搭建的 MIPS 微系统进行强测，同时添加一条新指令

题目每年都会发生变化，题意描述可能与原题有一定差异

本文的目的在于解释一下本就说得不是很明白的教程，然后谈一谈我的理解，并不一定完全正确，最终还是应当以教程、高老板 PPT 和《See MIPS Run Linux》为准



Test Blog



21%



## Re-awake

阿保剛

纯音乐, 请欣赏



Something

± Re-awake

阿保剛

## H<sub>2</sub>课下设计概述 #

要求实现的指令集为 **MIPS-C4** , 即 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, 在 P6 的基础上新增加了 MFC0、MTC0、ERET

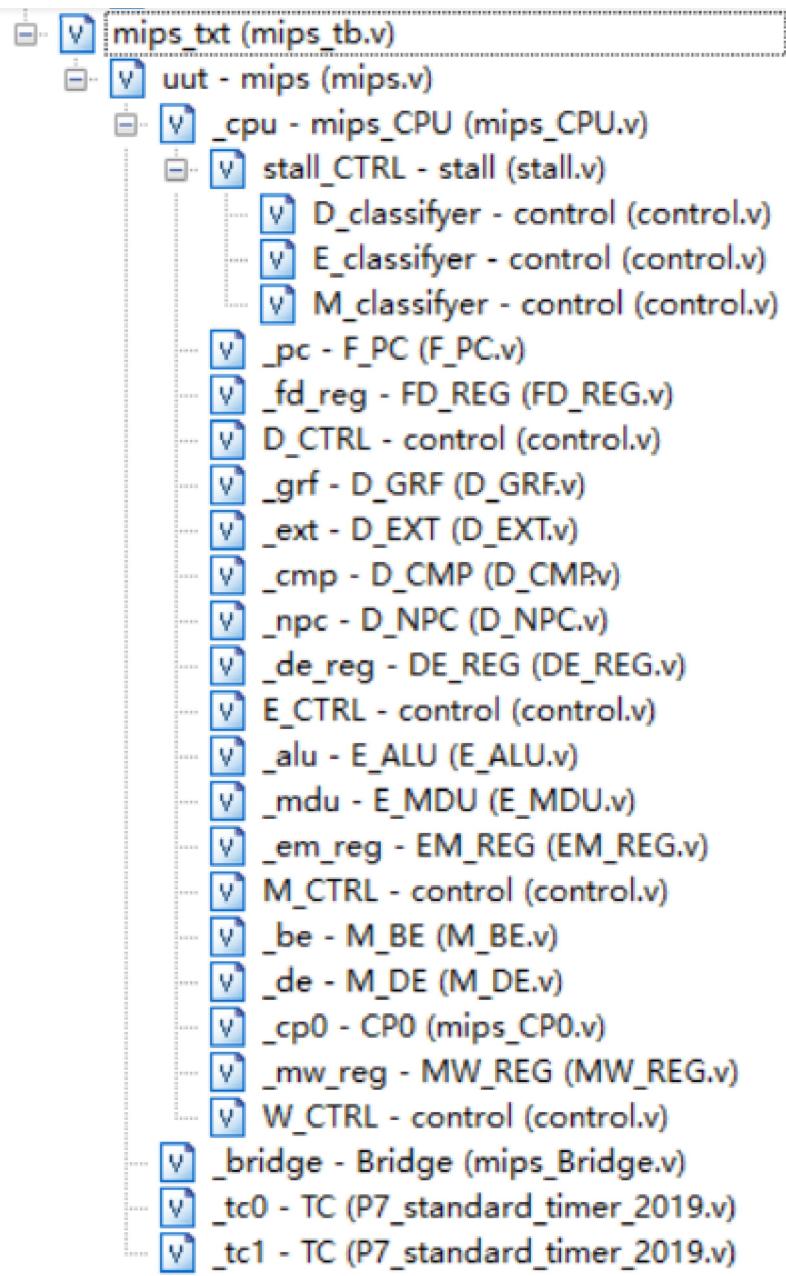
整体结构目录树如下:



Test Blog

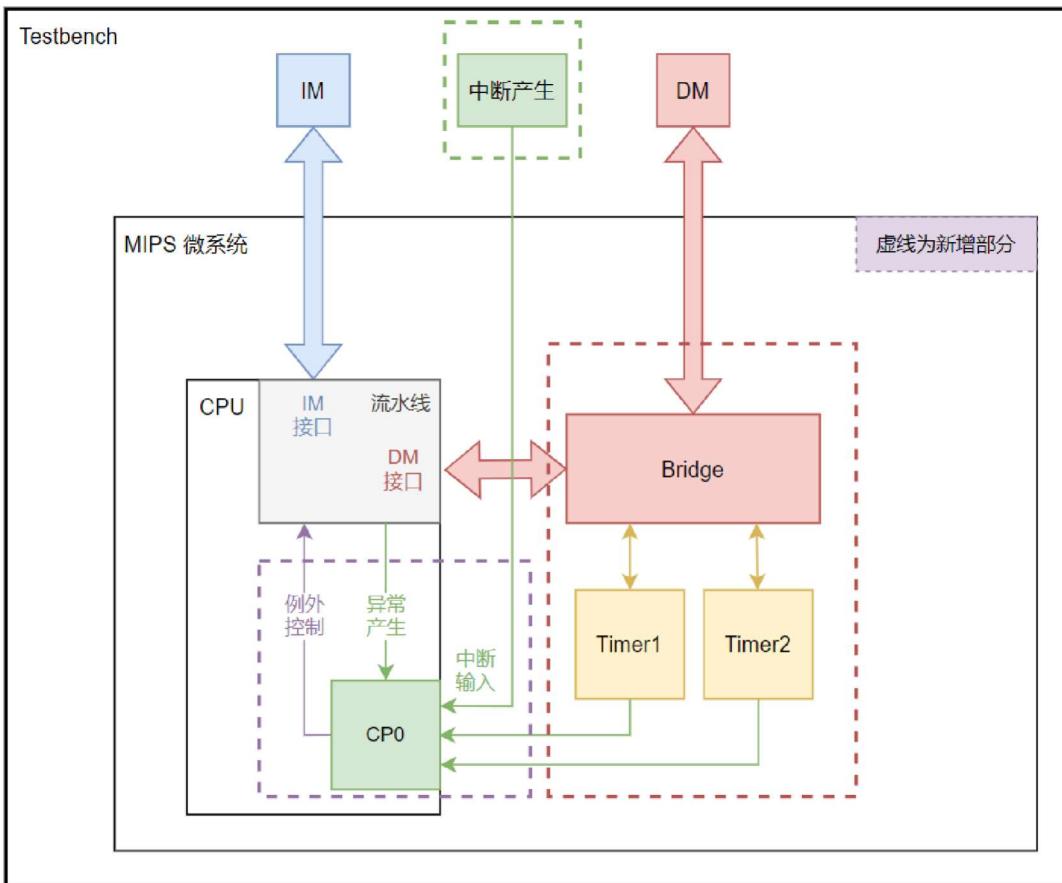


21%



MIPS 微系统整体设计如下图 (图片来源：2021 秋季计组教程)





P7 最难的就是课下，课下完成之后，课上就是三个提交窗口对课下进行强测，加一条指令，四过三即可

更难的是要读懂教程，因此这篇文章将尝试解读一下教程里面的内容

P7 主要还是要做三件事：**更改流水线各级使之可以产生异常，添加 CP0 处理异常，添加 Bridge 与两个外设（计时器）交互**，这三件事建议从前往后一件一件的做

首先对异常和中断的处理有一个整体的理解，可以理解成我们添加了**两种新的跳转指令**，第一种跳转的条件是**指令执行时产生异常或者外部、外设发出了中断信号**，跳转的位置是固定的 **0x4180**，至于怎么响应中断，怎么处理异常，从 0x4180 开始都是软件写好的，我们无需关心，跳转时要把相应的当前指令地址（**如果是延迟槽的话就是当前指令地址 - 4**）写入 EPC，第二种跳转条件是遇到 **eret** 指令，我们需要从当前位置跳回 EPC 中的地址，同时 \*\* **eret** 后面的指令不能被执行 \*\*

第一种情况需要更改流水线各级，使之可以产生异常，发生异常之后，我们把异常信号一直传递到 M 级的 CP0，CP0 负责决定是否接受并处理这个异常，如果 CP0 决定处理，这时就需要进行跳转行为



第二种情况对于 `eret` 的处理，我们是在 D 级判断，如果出现 `eret` 指令，就需要置位 F\_PC 和 NPC 分别为 EPC 和 EPC+4（想一想为什么要置位 NPC？）

对于异常的处理，按照教程，我们需要处理的异常包括下面几种

异常与中断码	助记符与名称	指令与指令类型	描述
0	<code>Int</code> (外部中断)	所有指令	中断请求，来源于计时器与外部中断
4	<code>AdEL</code> (取指异常)	所有指令	PC 地址未字对齐
4	<code>AdEL</code> (取指异常)	所有指令	PC 地址超过 <code>0x3000 ~ 0x6ffc</code>
4	<code>AdEL</code> (取数异常)	<code>lw</code>	取数地址未与 4 字节对齐
4	<code>AdEL</code> (取指异常)	<code>lh</code> , <code>lhu</code>	取数地址未与 2 字节对齐
4	<code>AdEL</code> (取指异常)	<code>lh</code> , <code>lhu</code> , <code>lb</code> , <code>lbu</code>	取 Timer 寄存器的值
4	<code>AdEL</code> (取指异常)	load 型指令	计算地址时加法溢出
4	<code>AdEL</code> (取指异常)	load 型指令	取数地址超出 DM、Timer0、Timer1 的范围
5	<code>AdES</code> (存数异常)	<code>sw</code>	存数地址未 4 字节对齐
5	<code>AdES</code> (存数异常)	<code>sh</code>	存数地址未 2 字节对齐
5	<code>AdES</code> (存数异常)	<code>sh</code> , <code>sb</code>	存 Timer 寄存器的值



异常与中断码	助记符与名称	指令与指令类型	描述
5	AdES (存数异常)	store 型指令	计算地址加法溢出
5	AdES (存数异常)	store 型指令	存数地址超出 DM、Timer0、Timer1 的范围
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

则可以发现我们要处理的部分包括 F 级的 IFU 部分，D 级的 Control 译码部分和 E 级的 ALU 部分以及**最难的 M 级的 DM 部分**

直接放出对应的代码块供参考

Verilog

```

1 // ----- IFU -----
2 wire [31:0] tmp_F_PC;
3
4 F_PC _pc(
5     .clk(clk),
6     .reset(reset),
7     .Req(Req),
8
9     .PC_WrEn(PC_WrEn),
10    .NPC(NPC),

```

可以看到关键在于对 F\_EXC\_AdEL 的赋值，为什么要向 PC 中传入 Req (中断请求) 呢？显然是因为上面的第二条，如果出现中断 PC 需要立刻变成 0x4180；此外，按照要求，如果出现异常，向后提交的 Instr 应当是 0，所以会有最后一句

对于 D 级的不认识的指令异常，我们直接在 Control 里面修改，如果有无法译码的指令，直接连出来一个信号表示异常



```

1 assign D_EXC_RI = !(beq | bne | bgez | bgtz | blez | bltz |
2                               j | jal | jalr | jr |
3                               lb | lbu | lh | lhu | lw | sb | sh | sw |
4                               lui | addi | addiu | andi | ori | xori | slti | slti
5                               add | addu | sub | subu | And | Nor | Or | Xor | ori
6                               sll | sllv | sra | srav | srl | srsv |
7                               ((opcode == 6'b000000) && (funct==6'b000000)) |
8                               div | divu | mfhi | mflo | mthi | mtlo | mult | mult
9                               mtc0 | mfc0 | eret);

```

有些实现方式可能需要考虑 sll 和 nop 的问题，但是我貌似就不用，这里提醒一下

对于 ALU 的溢出检测，由于我们需要同时检测算术溢出和取值地址计算溢出，因此有两个，我们传入 ALU 的信号需要包括应当检测哪种类型的溢出，因此输入端口的定义如下，前两个输入就是检测什么溢出，后两个输出表示是否检测到异常：

```

1 module E_ALU(
2     input ALUDM0v,
3     input ALUAri0v,
4
5     input [31:0] A,
6     input [31:0] B,
7     input [3:0] ALUOp,
8     output reg [31:0] C,
9     output EXC_Ari0v,
10    output EXC_DM0v
11 );

```

内部这样改，反正都差不多，直接放代码出来

```

1 wire [32:0] ext_A = {A[31], A}, ext_B = {B[31], B};
2 wire [32:0] ext_add = ext_A + ext_B;

```



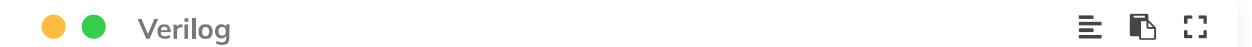
```

5          ((ALUOp == `ALU_add && ext_add[32] != ext_add[31])
6          ((ALUOp == `ALU_sub && ext_sub[32] != ext_sub[31]))
7 assign EXC_DM0v = (ALUDM0v) &&
8          ((ALUOp == `ALU_add && ext_add[32] != ext_add[31]
9          ((ALUOp == `ALU_sub && ext_sub[32] != ext_sub[31]))

```

M 级的检测主要是条件很多，需要对照着逐一检测，下面分别给出 BE 和 DE 里面检测储存和读取的异常检测

这个是储存的



```

1   wire ErrAlign = ((BEOp == `BE_sw) && (|Addr[1:0])) ||
2           ((BEOp == `BE_sh) && (Addr[0]));
3
4   wire ErrOutOfRange = !(((Addr >= `StartAddrDM) && (Addr <= `EndAddrD)
5           ((Addr >= `StartAddrTC1) && (Addr <= `EndAddrT
6           ((Addr >= `StartAddrTC2) && (Addr <= `EndAddrT
7
8   wire ErrTimer = (Addr >= 32'h0000_7f08 && Addr <= 32'h0000_7f0b) ||
9           (Addr >= 32'h0000_7f18 && Addr <= 32'h0000_7f1b) ||
10          (BEOp != `BE_sw && Addr >= `StartAddrTC1);
11
12 assign M_EXC_AdES = (store) && (ErrAlign || ErrOutOfRange || ErrTime

```

这个是读取的



```

1   wire ErrAlign = ((DEOp == `DE_lw) && (|Addr[1:0])) ||
2           ((DEOp == `DE_lh || DEOp == `DE_lhu) && (Addr[0]));
3
4   wire ErrOutOfRange = !(((Addr >= `StartAddrDM) && (Addr <= `EndAddrD)
5           ((Addr >= `StartAddrTC1) && (Addr <= `EndAddrT
6           ((Addr >= `StartAddrTC2) && (Addr <= `EndAddrT
7
8   wire ErrTimer = (DEOp != `DE_lw) && (Addr >= `StartAddrTC1);

```

宏自己定义去，我就不放了

现在我们就实现了下图中的检测功能



下面我们需要实现的是图中红线的功能，即异常码随流水线传递，直至提交到 CP0，**注意如果一条指令出现多个异常以先出现的为准**，具体在下面代码可以看出

```
1 // D 级
2 assign D_EXCCode = tmp_D_EXCCode ? tmp_D_EXCCode :
3                         D_EXC_RI ? `EXC_RI :
4                         `EXC_None;
5 // E 级
6 assign E_EXCCode = (tmp_E_EXCCode) ? tmp_E_EXCCode :
7                         (E_EXC_AriOv) ? `EXC_Ov :
8                         `EXC_None;
9
10 // M 级
11 assign M_EXCCode = (tmp_M_EXCCode) ? tmp_M_EXCCode :
12                         (M_EXC_AdES) ? `EXC_AdES :
13                         (M_EXC_AdEL) ? `EXC_AdEL :
14                         `EXC_None;
```

注意对于处于延迟槽中的指令，我们的 EPC 需要在 CP0 中特殊处理，因此当前指令是否在延迟槽内，我们需要新开一个信号 isInDelaySlot 跟着一起流水，另外对于阻塞插入的 nop，PC 值和 isInDelaySlot 都不正确，因此我们需要在 DE 流水线寄存器中特殊处理，利用下面的代码即可解决问题

```
1 if(flush || reset || Req || Stall) begin
2     E_PC <= Stall ? D_PC : (Req ? 32'h0000_4180: 32'd0);
3     E_Instr <= 32'd0;
4     E_rs_data <= 32'd0;
```

```

7      E_b_jump <= 0;
8
9      E_DelaySlot <= Stall ? D_DelaySlot : 0;
10     E_EXCCode <= 0;
11 end

```

我们既然已经实现了异常的检测与传递，下面就是 P7 异常处理的核心 CPO 处理器，它负责接受前面传来的异常信号和外部传入的中断信号，然后综合分析决定是否响应这个异常。

## H<sub>3</sub> CPO 协处理器

#

### 介绍

协处理器 0，包含 4 个 32 位寄存器，用于支持中断和异常。

### 端口定义

端口	输入 / 输出	位宽	描述
A1	I	5	指定 4 个寄存器中的一个，将其存储的数据读出到 RD
A2	I	5	指定 4 个寄存器中的一个，作为写入的目标寄存器
Din	I	32	写入寄存器的数据信号
PC	I	32	目前传入的下一个 EPC 值
ExcCodeIn	I	5	目前传入的下一个 ExcCode 值
isInDelaySlot	I	32	目前传入的下一个 BD 值
HWInt	I	6	外部硬件中断信号
WE	I	1	写使能信号，高电平有效
EXLClr	I	1	传入 eret 指令时将 SR 的 EXL 位置 0，高电平有效
clk	I	1	时钟信号
reset	I	1	同步复位信号



端口	输入 / 输出	位宽	描述
Req	O	1	输出当前的中断请求
EPCOut	O	32	输出当前 EPC 寄存器中的值
Dout	O	32	输出 A 指定的寄存器中的数据
TestIntResponse	O	1	检测 CPU 是否对外部中断产生响应，从而决定是否去写 0x0000_7f20

## 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将所有寄存器的值设置为 0
2	读数据	读出 A1 地址对应寄存器中存储的数据到 RD；当 WE 有效时会将 WD 的值会实时反馈到对应的 RD，当 ERET 有效时会将 EXL 置 0，即内部转发。
3	写数据	当 WE 有效且时钟上升沿到来时，将 WD 的数据写入 A2 对应的寄存器中。
4	中断处理	根据各种传入信号和寄存器的值判断当前是否要进行中断，将结果输出到 IntReq。

内部主要干两件事，**处理异常中断和管理那四个寄存器**（我们只需要实现其中 4 个就行，注意还有两个是只读的，写入应该忽略）

**请注意：不管教程和 PPT 写的是啥玩意，那四个寄存器每一个寄存器请务必开满 32 位！！！**

**否则课上会 GG**

管理寄存器就跟乘除槽管理 HI 和 LO 一样，开四个寄存器，然后根据 mfc0 和 mtc0 这两条指令处理他们的值就好

处理异常时需要干下面几件事：



将异常码 ExcCode、是否处于延迟槽中的判断信号 isInDelaySlot 和 \*\* 当前 PC (如果取指地址异常则传递错误的 PC 值) \*\* 一直跟着流水线到达 M 级直至提交至 CP0，由 CP0 综合判断分析是否响应该异常

如果需要响应该异常，则 CP0 输出 Req 信号置为 1，此时 FD、DE、DM、MW 寄存器响应 Req 信号，清空 Instr，将 PC 值设为 0x4180，然后输入 F 级的 NPC 也被置为 0x4180，下一条指令从 0x4180 开始执行

当外设和系统外部输入中断信号时，CP0 同样也会确认是否响应该中断，然后把 Req 置为 1，执行相同的操作。异常中断同时发生则中断优先！

当系统外部输入中断信号时，CP0 还会输出一个 TestIntResponse 信号指示是否响应外部中断信号，如果响应则系统会相应去写 0x7f20 地址，从而时外部中断信号停止（2021 年新增！）

还有一件事是关于乘除槽的，如果有异常在乘除槽之前被检出，那么就不执行乘除法，简单来说就是开启乘除槽条件在 Start 为 1 的基础上还要加一个 Req 为 0，这一点看一卡代码就行

如果是 `eret` 指令，那么 `EXL` 需要置 0，表明当前处于用户态，并没有在处理异常（这时候可以响应别的异常了），对于 eret 指令，我们还需要修改 NPC，因为不管是跳到异常处理程序还是跳回去都需要改变 NPC 的值，具体的改变化代码如下：

● ● Verilog



```
1 assign NPC = (Req) ? 32'h0000_4180 :  
2           (eret) ? EPC + 4:  
3           (NPCOp == `NPC_pc4) ? F_PC + 4 :  
4           (NPCOp == `NPC_jr_jalr) ? rs :  
5           (NPCOp == `NPC_b && b_jump) ? D_PC + 4 + <!!--swig@0-->, imm :  
6           (NPCOp == `NPC_j_jal) ? {D_PC[31:28], imm26, 2'b00} :  
7           F_PC + 4;
```

对于新加的两条处理 CP0 寄存器的指令，我没加任何转发，只是做了阻塞

只需要在 stall\_control 里面加一句话就行：



```
1     wire stall_ere = (D_ere) && ((E_mtc0 && E_rd_addr == 5'd14) || (M_mtc0 && M_rd_addr == 5'd14))
```

### H<sub>3</sub> Bridge #

系统桥是处理 CPU 与外设（两个计时器）之间信息交互的通道

CPU 中 store 类指令需要储存的数据经过 BE 处理后会通过 m\_data\_addr, m\_data\_byteen, m\_data\_wdata 三个信号输出到桥中，桥会根据写使能 m\_data\_byteen 和地址 m\_data\_addr 来判断到底写的是内存还是外设，然后给出正确的写使能

load 类指令则是全部把地址传递给每个外设和 DM 中，然后桥根据地址选择从应该反馈给 CPU 从哪里读出来的数据，然后 DE 在处理读出的数据，反馈正确的结果

Bridge 的端口列表如下：



这个很容易加的，直接根据地址选择是读内存还是外设，是传给 CPU 外设读出来的数据还是内存的数据

对于我的设计而言，CPU 仅仅与 Bridge 交互，输出地址、输出数据、输入的数据全部进入 Bridge 里面，然后 Bridge 会根据地址判断到底是往哪里写，又反馈给 CPU 什么数据

**不用担心越界的问题，这次的 DM 和 Timer 都是助教给的，他们已经考虑的很全面了**

### H<sub>3</sub> 顶层 Mips 微系统 #

最后 mips.v 中实例化 CPU, Bridge, TC0 和 TC1 三个模块相互交互

利用这样两句实现写 0x7f20，停止中断使能（2021 新增！）



### H<sub>2</sub> 课上测试说明 #

就是强测一下，按照我上面提到的注意点，根本不会出现任何问题

加的那条指令描述如下：



`tltiu $t0, imm16`，如果 GPR [rs] < imm16，那么会产生 Trap (EXCCode = 13) 的异常，imm16 符号扩展

解法如下

加一下 Trap 异常，就是在 def.v 里面定义一下

决定在哪里处理异常，反正我是在 E 级，然后把它放进 calc\_i 类指令里，自动转发阻塞支持

在 ALU 里面加一句判断，多连出来一个异常指示信号

**重点在这里：我们已经把它归入 calc\_i 类，但是这条指令不应当写寄存器，因此在要写的寄存器编号那里需要特判一下，把这条指令要写的寄存器特判为 0**

然后就搞定了

## H<sub>2</sub> 思考题参考答案

#

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？)

“硬件 / 软件接口”是指令（机器码）。硬件实现了一些功能，并按照规约可以被相应的指令所操控。软件通过规约使用相应的指令操控硬件完成相应的功能，从而达到软件所期望的效果。指令在这个过程中实现了硬件软件的对接，因此是“硬件 / 软件接口”。

2 BE 部件对所有的外设都是必要的吗？

不是，只有对字节 / 半字有存取需求的才有必要。

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。

见计时器说明文档。

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 $12, 0x0401  
mtc0 $12, $12  
li $1, 100  
li $2, 9  
sw $1, 0x7f04($0)  
sw $2, 0x7f00($0)  
  
dead_loop:  
j dead_loop  
nop  
  
.ktext 0x4180  
li $1, 100  
li $2, 9  
sw $1, 0x7f04($0)  
sw $2, 0x7f00($0)  
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？

鼠标和键盘产生中断信号，进入中断处理区的对应位置，将输入信号从鼠标和键盘中读入寄存器。

[北航计算机组成原理](#) [Verilog](#) [MIPS](#) [流水线CPU](#) [P7课上](#) [P7课下](#)

更新于 2021-12-15 阅读次数 1773 次

赞赏

请我喝[茶]~(￣▽￣)~\*



Test Blog



21%

① 本文作者：FL @ FlyingLandlord's Blog

② 本文链接：<http://flyinglandlord.github.io/2021/12/15/BUAA-CO-2021/P7/P7课上&课下/>

③ 版权声明： 本站所有文章除特别声明外，均采用 ④ BY-NC-SA 许可协议。转载请注明出处！

上一篇

▶ 北航计算机组成原理

## P6课下&课上总结

下一篇

▶ 北航计算机组成原理

## P8课下&课上总结





Test Blog



21%



## 最新评论

1 KARLON @ 2 days ago

P7过不过都得给您磕一个

2 Weixinvcci @ 6 days ago

P7过了高低得给您磕一个

3 Jew @ 7 days ago

P7过了高低得给您磕一个orz

4 StrivingLee @ 8 days ago

P7过了高低得给您磕一个orz

5 hihiczx @ 10 days ago

P7过了高低得给您磕一个orz

6 fysszlr @ 11 days ago

P7过了高低得给您磕一个

7 花露水 @ 11 days ago

P7过了高低得给您磕一个Orz<img src="https://cdn.jsdelivr.net/gh/MiniValine/tieba@master/tieba-3...

8 pigKiller @ 15 days ago

P7过了高低得给您磕一个

9 Harry Potter @ 16 days ago

P7过了高低得给你磕一个



10 花露水 @ 80 days ago

大佬保佑我今晚pre不挂

---

## 随机文章

1 北航计算机组成原理

Pre课上测试游记

---

2 北航操作系统

BUAA-OS Lab5实验总结

---

3 北航计算机组成原理

P3课下学习—单周期CPU设计(1)

---

4 北航计算机组成原理

CPU输出序列检查调试方法

---

5 北航计算机组成原理

P7课下&课上总结

---

6 北航计算机组成原理

Verilog Pre考前复习

---

7 北航操作系统

BUAA-OS Lab0实验总结

---

8 北航操作系统

BUAA-OS Lab4实验总结

---

9 北航操作系统

BUAA-OS Lab3上机

---

10 北航计算机组成原理

MIPS汇编哈密顿回路

---

© 2010 – 2022  FL @ Test Blog

基于 Hexo & Theme.Shoka

