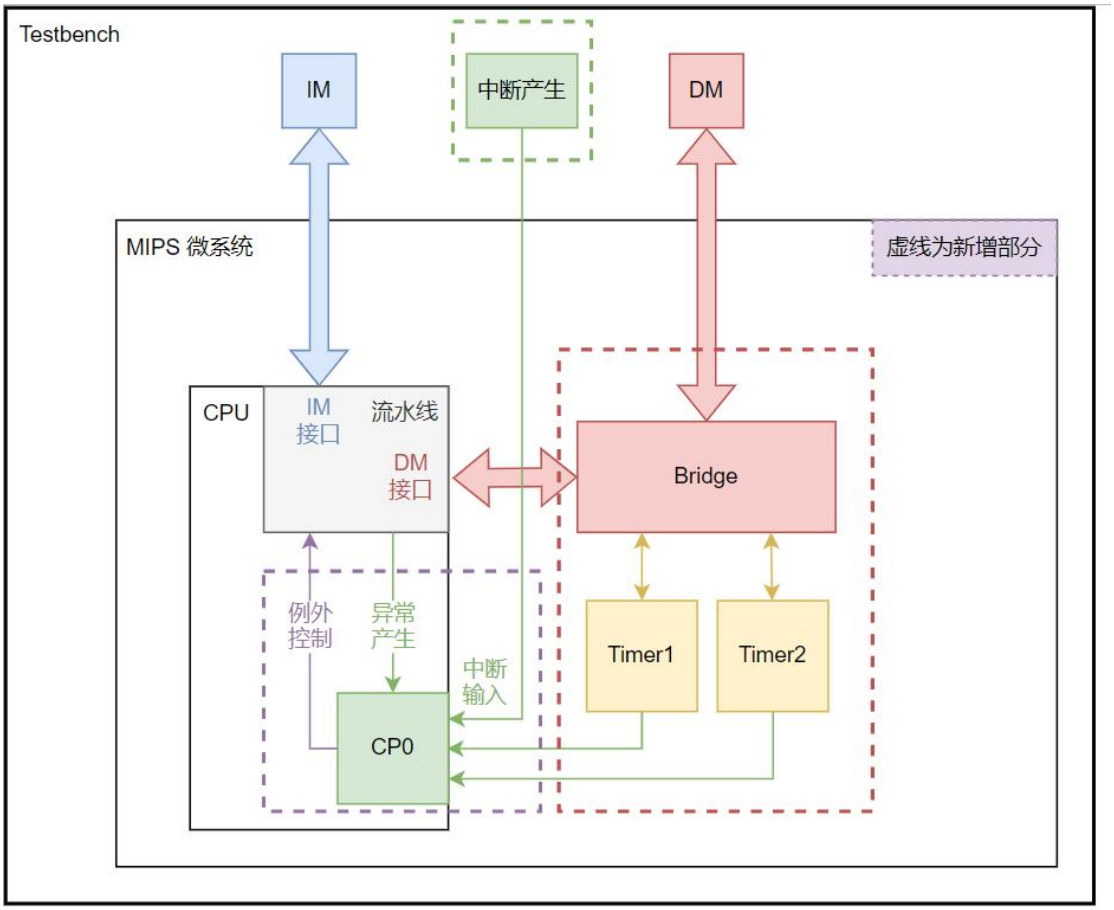
一、 计算机组成原理实验报告参考模板

1.1 一、CPU 设计方案综述

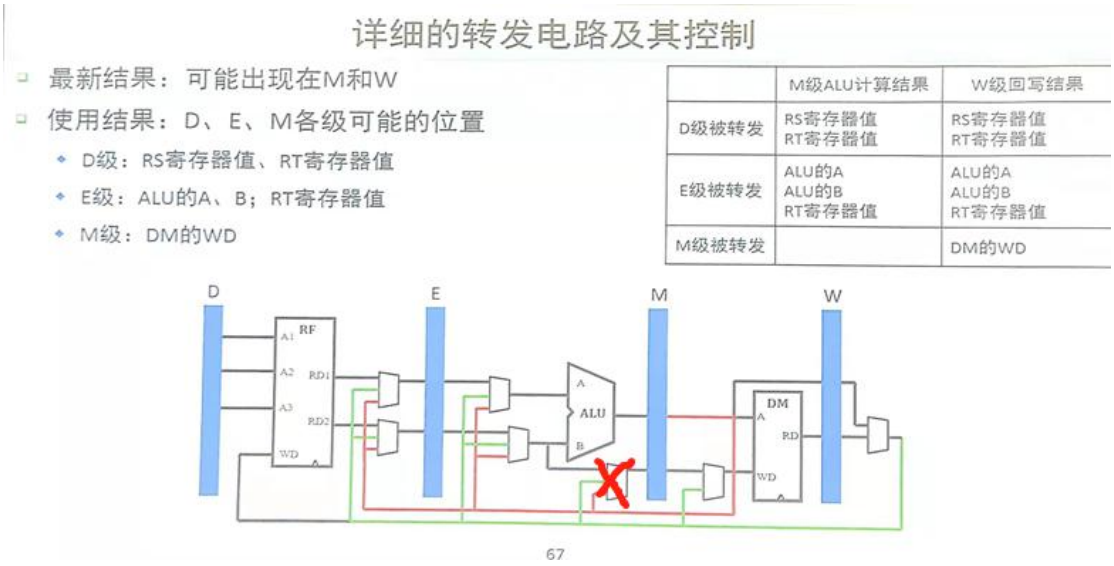
1.1.1 （一）总体设计概述

我们设计的 CPU 将包含 Controller（控制器）、IFU（取指令单元，我使用了三个单元 PC, NPC，IM 来实现）、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）、ALU（算术逻辑单元）、DM（数据存储器）、 EXT（位扩展器）等基本部件，通过 MUX、 Splitter 等内置器件组合连接成数据通路。并采用分布式的 Control Unit, stall ctrl, forward ctrl 对数据通路进行控制



1.1.2 （二）关键模块定义

1.1.3 数据通路总览&示意图



CP0

```
// R&W like grf,to change EPC
input [4:0] A,
input [31:0] WD,
input WE,
output [31:0] RD,
//detect interrupt
input [6:2] exc_code,
input [7:2] HW_int,

//save if it is for EPC
input [31:0] PC,
input BD,
//epc reg out
output [31:0] EPC_RD,
//when detect exc or int, the pipeline need change
output int_exc_req,
output intt_req,
input EXL_clr, //excp return
input clk,
input reset
```

BRG

```
input [31:0] PrAddr,
output [31:0] PrRD,//
input [31:0] PrWD,
input PrWE,
output [31:0] DEV_Addr,//
```

input [31:0] DEV0_RD,
input [31:0] DEV1_RD,
output DEV0_WE,//
output DEV1_WE, //
output [31:0] DEV_WD

1.2 F 级

1.2.1 PC 同 P5 略

1.2.2 IM 外置略

1.3 D 级

1.3.1 GRF 同 P5 略

（可使用表格进行端口说明）

1.3.2 NPC 同 P5 略

1.3.3 EXT 同 P5 略

1.3.4 CMP 同 P5 略

1.3.5 D|E|M|W 级分布式 ctrl

,//每个指令确定后控制信号就都定了
,//
,

IO	Description
input [31:0] instr,	
input [1:0] cmp,	比较信息
input [1:0] zero,	
input stall	当前是否处于 stall
input [4:0] DM_RD	确定 A3 用的
output[2:0]fwd_src_sel	转发源选择
output reg_write	同普通控制功能
output [2:0]reg_dst	
output[2:0]which_to_reg	
output [1:0] ALU_src	
output mem_write	
output [2:0]NPCop	顾名思义
output [4:0]ALU_op	
output [4:0]MDU_op	
output start	MDU 的 start 信号
output CAL_sel	选择 MDUorALU
output [1:0]LS_op	DM 的符号和取数信息

output DM_sign	
output sign	EXT 的扩展控制
output Tuse_rs_0	每个指令固有的属性，丢给 forward unit 来控制转发
output Tuse_rs_1	
output Tuse_rt_0	
output Tuse_rt_1	
output Tuse_rt_2	
output [2:0] new_at	每个指令确定后，产生在哪里确定
output [4:0]A1	每个指令确定，他们的操作对象确定
output [4:0]A2	
output [4:0]A3	
output null_slot	清空延迟槽
output islrm	判断特殊的访存指令

1.4 E 级

1.4.1 ALU 同 P5 略

1.4.2 MDU

IO	Description
input start	开始
input [31:0] D1	**输入数据 1**
input [31:0] D2	输入数据 2
input [4:0] MDU_op	
input clk	
input reset	
output real_busy	真正的 busy = busy | start
output [31:0] MDU_out	**计算结果**

1.5 M 级

1.5.1 DM 外置

1.5.2 BYTEEN

IO	Description
input [31:0] A	读取地址

input WE	写入 DM 信号
input [1:0]LS_op	Load_store op
output reg[3:0] byte_en	Enable 哪个 byte(one-hot)

1.5.3 DATAEXT

```
input [31:0] A,
input [31:0] RDin,
input [1:0] LS_op,
input DM_sign,
output reg[31:0] RDout
```

IO	Description
input [31:0] A	读取地址
input [31:0] RDin	读入字
input [1:0]LS_op	Load_store op
input DM_sign	对读入的字进行截取后是否符号扩展
output reg[31:0] RDout	处理后的读出数据

1.6 W 级

1.7 FORWARD CONTROL

见 P5 略

1.8 STALL CONTROL

见 P5 略

1.8.1 （三）真值表与数据通路控制表

最初版真值表

	op	func	Regwrite	RegDst	ALUsrc	Branch	MemWrite	MemtoReg	sllv?
add	0	100000	1	b01	0	0	0	0	
sub	0	100010	1	b01	0	0	0	0	
and	0	100100	1	b01	0	0	0	0	
or	0	100101	1	b01	0	0	0	0	
sll	0	0	1	b01	0	0	0	0	0use instr shamt
sllv	0	100	1	b01	0	0	0	0	1use RD1
slt	0	101010	1	b01	0	0	0	0	
jr	0	1000	0	x1	0	0	0	0	
jalr(x)	0	1001	1	b10($31)					
j	10		0	x0	0	0	0	0	
jal	11	0	1	b10($31)	0	0	0	0	
addi	1000		1	b00	1	0	0	0	
addiu(x)	1001		1	b00	1	0	0	0	
ori	1101		1	b00	1	0	0	0	
lui	1111		1	b00	1	0	0	0	
slti	1010		1	b00	1	0	0	0	
sw	101011		0	x0	1	0	1	0	
lw	100011		1	b00	1	0	0	1	
beq	100		0	x	0	1			
blez(x)	110								

重新改版：

module	NPC							IM	EXT		RF					ALU				DM			
input	b	j	jr	aluz	imm26	ext_imm16	jreg	PC	sign	imm16	A1	A2	A3	WD3	A	B	shamt	ALUop	A	WD	sign	WE
addu								PC.PC			IM[25:21]	IM[20:16]	IM[15:11]	ALU.res	RF.RD1	RF.RD2		add				0
subu(同上												t										
beq	1			ALU.zero		EXT.imm		t	CTRL1	IM.instr	IM[25:21]	IM[20:16]			RF.RD1	RF.RD2		sub()eq				
lw								PC.PC	CTRL1	IM.instr	IM[25:21]	base)	IM[20:16]rt	DM.RD	RF.RD1	EXT.imm		add	ALU.res		0	0
sw								PC.PC	CTRL1	IM.instr	IM[25:21]	IM[20:16]rt			RF.RD1	EXT.imm		add	ALU.res	RF.RD2		1
jal		1			IM.instr			PC.PC					0x1f	PC.PC+4								
jr			1				RD.RD1	PC.PC			IM[25:21]											
j		1			IM.instr			PC.PC														
lb																						
sb																						

转发表

从reg转发到stage内部去				产生atEREG（存在EREG待写入的imm32,PC）			保留or产生atMREG					保留or产生atWreg						冲突的
供给需求重复就转发,管他这个时候要不要呢				jal	jalr	lui	cal_R	cal_I	jal	jalr	lui	cal_R	cal_I	jal	jalr	lui	lw	可以转
产生完了，往后传，留着以后转发				31	rd	rt	rd	rt	31	rd	rt	rd	rt	31	rd	rt	rt	待写入
需求atDstage(NPC，CMP)				转发的内容														
使用reg	需求指令	转发mux_ctrl	是否转发信号							但是RD1,RD2后面要用也要传下去								
rs	beq,jr,jalr	Dmux_fwd_rs	D_fwd_rs	EPC8	EPC8	Eext_imm32	MALU_res	MALU_res	MPC8	MPC8	Mext_imm32	WALU_res	WALU_res	WPC8	WPC8	Wext_imm32	WRD	
rt	beq	Dmux_fwd_rt	D_fwd_rt	EPC8	EPC8	Eext_imm33	MALU_res	MALU_res	MPC8	MPC8	Mext_imm33	WALU_res	WALU_res	WPC8	WPC8	Wext_imm33	WRD	
				冲突的、可以转发的指令们所处的位置			保留or产生atMREG					保留or产生atWreg						
需求atEstage(ALU, Mreg)				可以转发的流水线深处的指令们			cal_R	cal_I	jal	jalr	lui	cal_R	cal_I	jal	jalr	lui	lw	
使用reg	需求指令	转发mux_ctrl	是否转发信号	待写入的寄存器			rd	rt	31	rd	rt	rd	rt	31	rd	rt	rt	
rs	cal_R, cal_I,lw,sw	Emux_fwd_rs	E_fwd_rs				MALU_res	MALU_res	MPC8	MPC8	Mext_imm32	WALU_res	WALU_res	WPC8	WPC8	Wext_imm32	WRD	
rt	cal_R,	Emux_fwd_rt	E_fwd_rt				MALU_res	MALU_res	MPC8	MPC8	Mext_imm33	WALU_res	WALU_res	WPC8	WPC8	Wext_imm33	WRD	
rt	sw	Emux_fwd_rt2	E_fwd_rt															
				冲突的、可以转发的指令们所处的位置								保留or产生atWreg						
需求atMstage(DM_WD)				可以转发的流水线深处的指令们								cal_R	cal_I	jal	jalr	lui	lw	
使用reg	需求指令	转发mux_ctrl	是否转发信号	待写入的寄存器								rd	rt	31	rd	rt	rt	
rt	sw	Mmux_fwd_rt	M_fwd_rt									WALU_res	WALU_res	WPC8	WPC8	Wext_imm33	WRD	

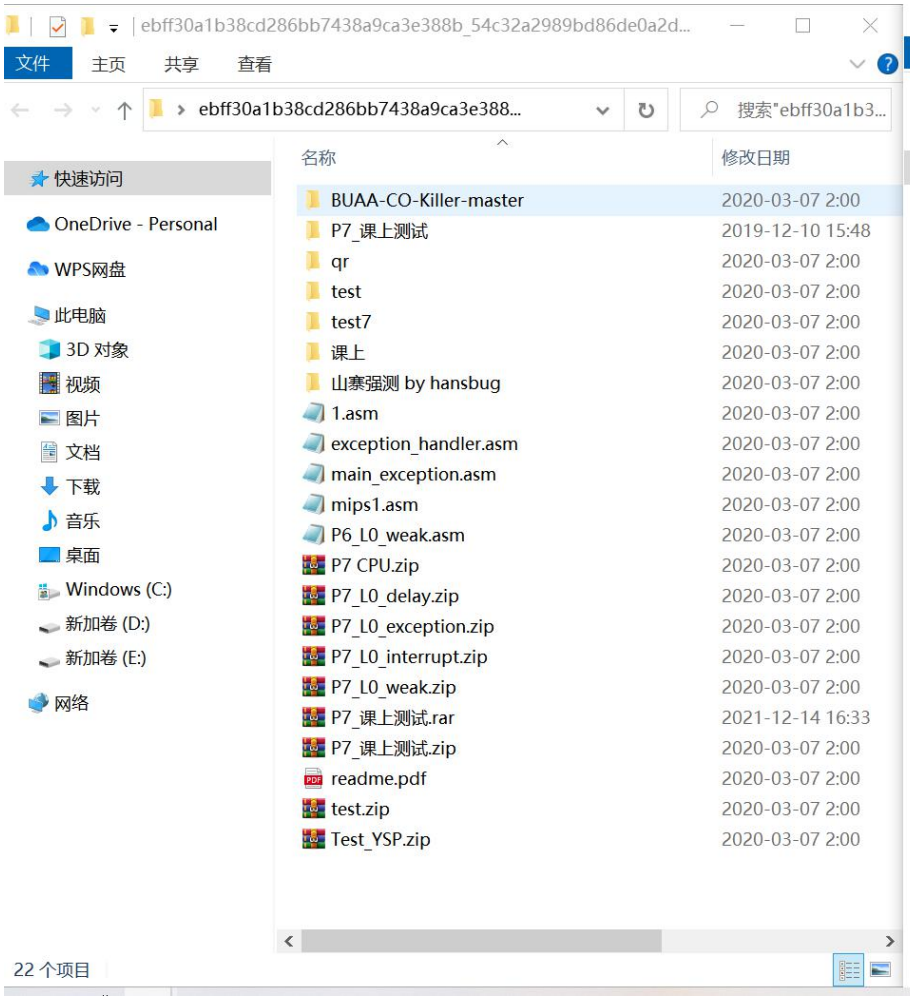
暂停表

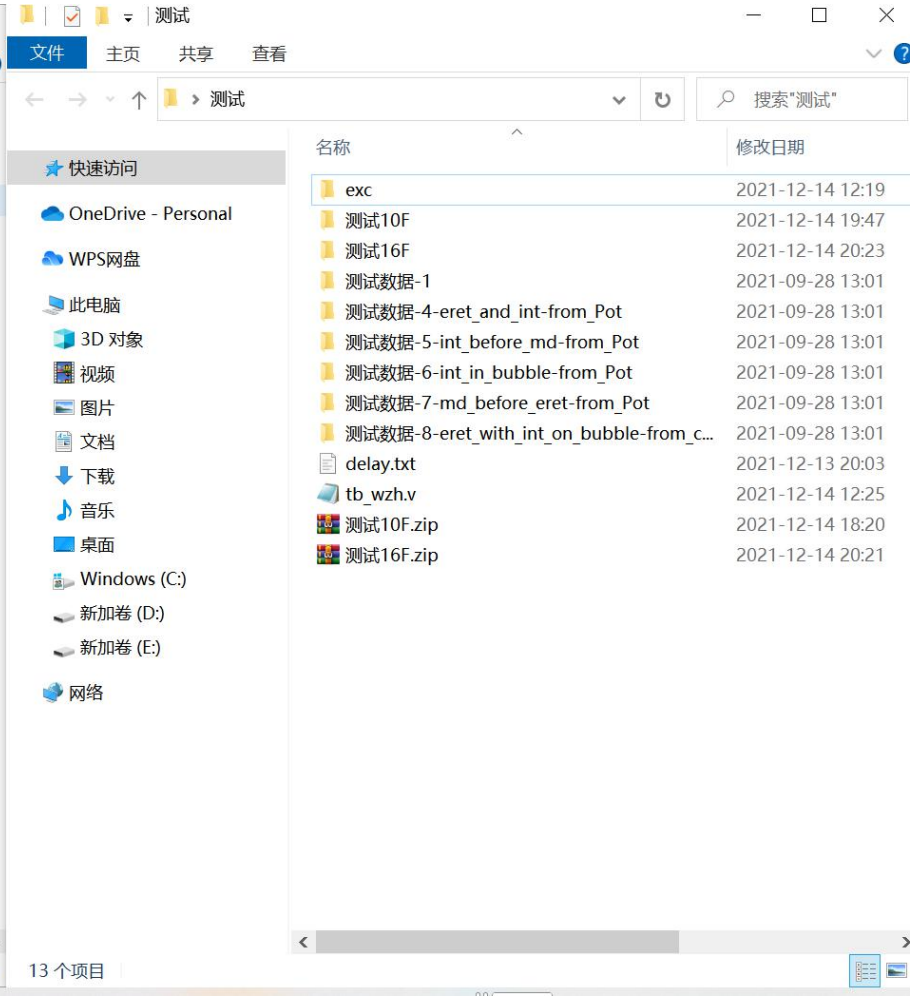
处于D级，判断是否stall				cal_R	cal_I	lw	jal	lui	jalr	cal_R	cal_I	lw	lw
若stall，下个上升沿，保持D,不变，清空E			结果产生在	M_Alures	M_alures	W_RD	E_imm	E_imm	E_imm	M_Alures	M_alures	W_RD	W_RD
PC保持			Tnew(E-res)	1	1	2	0	0	0	0	0	1(new at dm)	0
REGD.en=0 ID/EX.reset=1 PC.en=0			save at	rd(new at ALU)	rt(new at ALU)	rt(new_at_DM)	31	rt	rd	rd	rt	rt	rt
D				E						M			W
need/src	冲突aka需求寄存器	when	Tuse(D-use time)										
cal_R	rt/rs	E_ALU	1			stall							
cal_I	rs	E_ALU_A	1			stall							
beq	rs/rt	D_CMP	0	stall	stall	stall	D需求可以向D转发，见sheet3					stall	
lw	rs	E_ALU_A	1			stall							
sw	rs	E_ALU_A	1			stall							
sw	rt	M_WD	2										
jr	rs	F_NPC	0	stall	stall	stall	D需求可以向D转发，见sheet3					stall	
jalr	rs	F_NPC	0	stall	stall	stall						stall	

1.9 二、测试方案

重点在于测试 CP0,和 eret,mtc0mfc0 的阻塞转发，以及进入退出 kernel 时的关键情形

主要手段为和同学对拍，一些样例





1.9.1 （二）自动测试工具

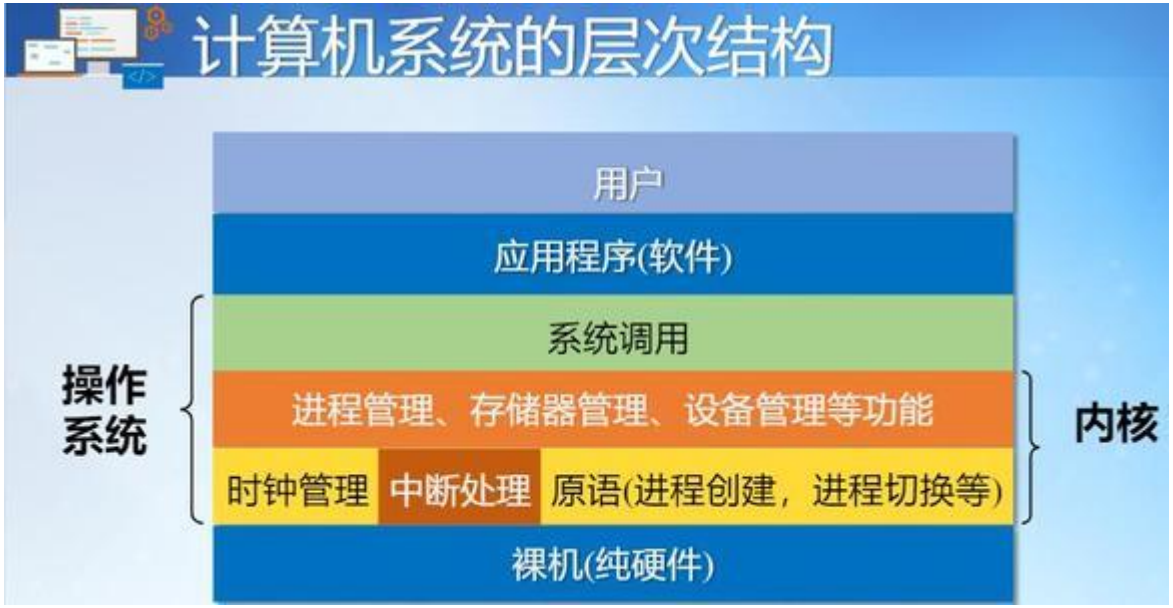
没来得及写

1.10 三、思考题

思考题汇总

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？)

硬件接口指的是硬件间的数据沟通， 软件、硬件间的接口正是通过流水线 mips 来实现的，软件通过编译给出汇编指令使得硬件可以识别，并且执行软件， 软件硬件协同工作；



引自百科：

接口（硬件类接口）是指同一计算机不同功能层之间的通信规则称为接口。[1]

接口（软件类接口）是指对协定进行定义的引用类型。其他类型实现接口，以保证它们支持某些操作。接口指定必须由类提供的成员或实现它的其他接口。与类相似，接口可以包含方法、属性、索引器和事件作为成员。

2、 BE 部件对所有的外设都是必要的吗？

应该不需要， 比如这次的 IM 就没有经过 bridge

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。

共同点：计数器使能有效时，每个周期值 cnt-1；减至零时中断；

重置时恢复为 preset 寄存器中的初始值。

不同点：模式 0 进入中断后将计数器使能置 0，直到计数器使能置 1 时结束中断；模式 1 进入中断后下个周期结束中断，再下一个周期恢复初始值。

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
   addi $1, $0, 9
   sw $1, 0x7f00($0) #ctrl IMask = 1,  enable = 1, mode: 00 1001


   addi $1, $0, 100
   sw $1, 0x7f04($0) #SR
   addi $1, $0,
    # IM[7:2] = 111111, SR[1] = EXL = 0, IE = 1
   mtc0 $1, $12
.ktext 0x4180
   addi $1, 9
   sw $t0, 0x7f00
   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？

鼠标点击、位置移动或按下键盘时，将产生中断信号，从而使 CPU 进入相应的中断处理程序。

二、系统桥与计时器

三、 系统桥 (bridge)

一个 MIPS 微系统不能仅由 CPU 构成，其还应该含有外部设备，这也就要求 CPU 有与不同外部设备进行沟通的能力。因此，我们需要实现与外部设备沟通的**系统桥 (bridge)**。在本次实验中，我们采用划分地址空间的方式来选择与哪个外设通信。下图是规定的地址空间设计，测试程序也将以此为根据编写。需要注意的是，本 project 与《See MIPS Run Linux》和 PPT 中给出的 MIPS 系统地址范围是不同的，而与 MARS 相同，这主要是为了能够让你能更好的验证设计。

	地址或地址范围	**备注**
数据存储器	0x0000_0000 至 0x0000_2FFF	
指令存储器	0x0000_3000 至 0x0000_6FFF	
PC 初始值	0x0000_3000	
Exception Handler 入口地址	0x0000_4180	
计时器 0 寄存器地址	0x0000_7F00 至 0x0000_7F0B	计时器 0 的 3 个寄存器

	地址或地址范围	备注
计时器 1 寄存器地址	0x0000_7F10 至 0x0000_7F1B	计时器 1 的 3 个寄存器

最后两行是为我们此次实验的外部设备之一，计时器 (timer) 准备的。本页面的第二小节对计时器进行了详细介绍。

在 P7 实验中有两类外部设备，一类是存储器，一类是计时器。两类外设都通过系统桥模块与 CPU 相连。我们可以把系统桥理解为一个用于区分地址的组合逻辑模块，当读写计时器相关地址时，CPU 通过系统桥与我们自己实例化的两个 timer 进行数据交互；当需要读写数据存储器时，CPU 通过系统桥将相关信号上传至顶层模块（`mips.v`），与官方 tb 中实现的存储器进行数据交互。

关于系统桥的具体编写，请大家参考该文件 L15-支持 IO.pdf。

独立的系统桥

在实现系统桥时，其必须作为独立 module 来实现，不能包含在 CPU 内部。

四、 计时器 (timer)

在 P7 这个简单的 MIPS 微系统中，计时器是一种外部设备，其主要功能就是**根据设定的时间来定时产生中断信号**，是我们系统的中断来源。

怎样使外设与 CPU 进行沟通呢？采用划分地址空间的办法后，与外设沟通和与 DM 沟通的方式类似，通过一个 CPU 视图下的内存地址，读写相应数据达到与外设沟通的目的。而这个所谓的内存，在外设中，实际上只是若干寄存器。系统桥传入对地址的访问请求后，我们通过计时器内部的转换代码，将请求转变为对相应寄存器的读写操作。

除了使用访存请求来沟通外，CPU 还需要响应外设和 Testbench 传入的中断信号，并执行对应的中断处理程序。

往届同学在完成 P7 时，教程通过文字叙述与转移图等尽可能解释计时器行为规范，但依旧存在许多争议点，因此最终实现的计时器在功能与操作上存在细微的差异。因此在今年的教程中，我们决定向同学们提供已实现的计时器 Verilog 源代码： 源代码。 timer 内部需要定义多个程序员可见寄存器，如 **CTRL**、**PRESET** 等，也需要定义若干用于完成功能的内部寄存器(程序员不可见)，详情请参考设计文档：CO 定时器设计规范-1.0.0.4.pdf

同学们需要在阅读源代码后，完成以下内容并以 PDF 文档的形式进行提交：

1. 请用**状态转移图**的方式分别描述两种模式下计时器状态和状态转移的条件。有关状态转移图的规范和绘制工具，可参考 ProcessOn 或 diagrams.net。
2. 编写**计时器使用说明**，分别阐述不同计时器模式中、不同计时器状态下，用户对计时器的操作规范(即针对计时器寄存器的操作)，规范中应包括可行的操作与对应功能、不可行操作与误操作后果等等。

思考题

1. BE 部件对所有的外设都是必要的吗？
2. 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。异常与中断

五、 异常与中断

同学们可能在之前的程序设计学习过程中接触过异常这个概念， Python、Java 等语言中都有异常处理的机制。而我们现在所谈论的是与上述“异常”有所不同的东西——**硬件异常**，比如加法**溢出**，乘除法**除零**这些不在原本部件设计范围内的运算发生后， CPU 就会产生一个硬件异常。而所谓的中断，更多情况是来自外部，例如计时器、输入设备（更加详细准确的定义请同学们参考相应的资料）、存储设备等。从这个角度出发，在笔者看来中断比起异常是更为**宽泛**的，我们可以把异常看做是一种 CPU 自身产生的中断（内部中断），因此相应的 CPU 就能获得更多异常相关的信息。**下文中为方便表述将异常与中断统称为中断**。总的来看，异常和中断就是 CPU 因为某种内部事件或者外部事件，正常运行的运行进程被中止，需要进行相关的处理。

六、 精确异常

对于硬件异常，我们能明确指出是哪条指令导致了异常，并称这条指令为异常受害指令。精确异常的特性是，在异常受害指令**前面的所有指令都执行完毕**，而**受害指令及其后续指令都像从来没有开始**（准确说是当异常处理结束后重新执行这些指令，与未发生异常时执行这些指令的效果一样）。这样的处理思路使得从使用者的角度来看， CPU 执行异常处理是顺序执行的，而隐藏了流水线设计的细节。

七、相关指令

从这个角度看，引入中断和异常机制后，我们就拥有了一个**有多种状态**的 CPU ，一种是正常流水状态，一种是中断处理状态，为了对此进行相关的控制，我们需要引入新的部件 **CP0（Coprocessor 0）**， CP0 的具体功能与设计我们在此先按下不表。而拥有了 CP0 后的 CPU 就能够支持如下的指令（详情请参阅指令集）：

- MTC0：通用寄存器值写入 CP0 寄存器
- MFC0：读取 CP0 寄存器至通用寄存器
- ERET：从中断中返回

细心的同学可能会发现，其中并没有进入中断状态的指令，这是由于中断产生时，我们的 CPU 会直接改写 PC 至中断处理程序所在地址，自主进入中断处理程序。然后通过 MFC0 、 MTC0 等指令从 CP0 中获取相关信息，设置 CPU 状态，进行相关处理后，通过 ERET 返回正常状态。

八、中断处理程序

前文已叙，当 CPU 遇到中断时，会改写相应 PC，进入中断处理程序。而中断处理程序的大致框架如下：

软件实现：中断服务程序

- 框架结构：保存现场、中断处理、恢复现场、中断返回
- 1、保存现场
  - 将所有寄存器都保存在堆栈中
- 2、中断处理
  - 读取特殊寄存器了解哪个硬件中断发生
  - 执行对应的处理策略（例如读写设备寄存器、存储器等）
- 3、恢复现场
  - 从堆栈中恢复所有寄存器
- 4、中断返回
  - 执行eret指令

1、3、4：通用
2：针对特定设备

所谓**保存现场**，即是将当前 CPU 运行状态给保留下来，通过之前对函数调用的学习我们可以知道，CPU 运行状态中最重要的就是寄存器的状态，我们需要在中断处理程序中把所有寄存器存入堆栈，在返回时再予以恢复，以此来达成对状态的恢复。同时我们需要注意的是，在中断处理程序中，我们可能需要对 CP0 中寄存器进行操作，请大家参阅《See MIPS Run Linux》了解更多信息。另外，针对外部中断，我们也需要对外部硬件进行操作。

中断处理程序为软件实现，不需要同学们在 P7 实现，仅作为了解即可。

九、 中断和返回

那么，在中断/异常产生时，我们的 CPU 都要做哪些事情呢？

简单来说，在中断/异常产生时，CPU 会跳到预先设定好的异常处理地址。这个地址就像 CPU 执行从 0x3000 开始一样，是 CPU 的根本属性。在我们的 CPU 中，这个地址是 0x4180。一旦 CP0 判断进入中断、异常，PC 就跳转至 0x4180。

此外，还要将发生中断/异常时的 PC 写入 CP0 中的 EPC 寄存器，并设置一些 CP0 中的寄存器，比如设置中断异常种类等。具体信息请参考 CP0 协处理器设计约束。中断处理程序会根据此时填入的信息，进行相应的处理。

特别地，由于我们的 CPU 是不支持异常重入的，在 CPU 处于异常处理状态时，不能再继续响应中断信号。当然，CPU 状态这个信息，也是记录在 CP0 的寄存器中的。

在异常返回时（ERET 指令），CPU 会跳到 EPC 记录的地址，并修改 CP0 中的若干状态信息。值得一提的是，中断处理前后的 PC 可能并不相同，因为中断处理程序可能会更改 EPC 的值。同样的，通用寄存器的值在异常处理前后也有可能不同。具体保存哪些寄存器，恢复哪些寄存器，都是由软件来决定的。

十、 利用 MARS 验证中断处理框架

尽管在 MARS 中，我们只能针对异常进行模拟，无法模拟外部中断。但由我们对异常与中断的了解可以知道，两者的处理是类似的。因此你可以在 MARS 中先验证你的中断/异常处理框架是否正确。方法是：你让某条指令出错（溢出等），然后看 MARS 能否进入你写的 exception handler。至于你如何处理这个错误，则是次要问题。

你可以按照下图的方式添加 Exception handler：

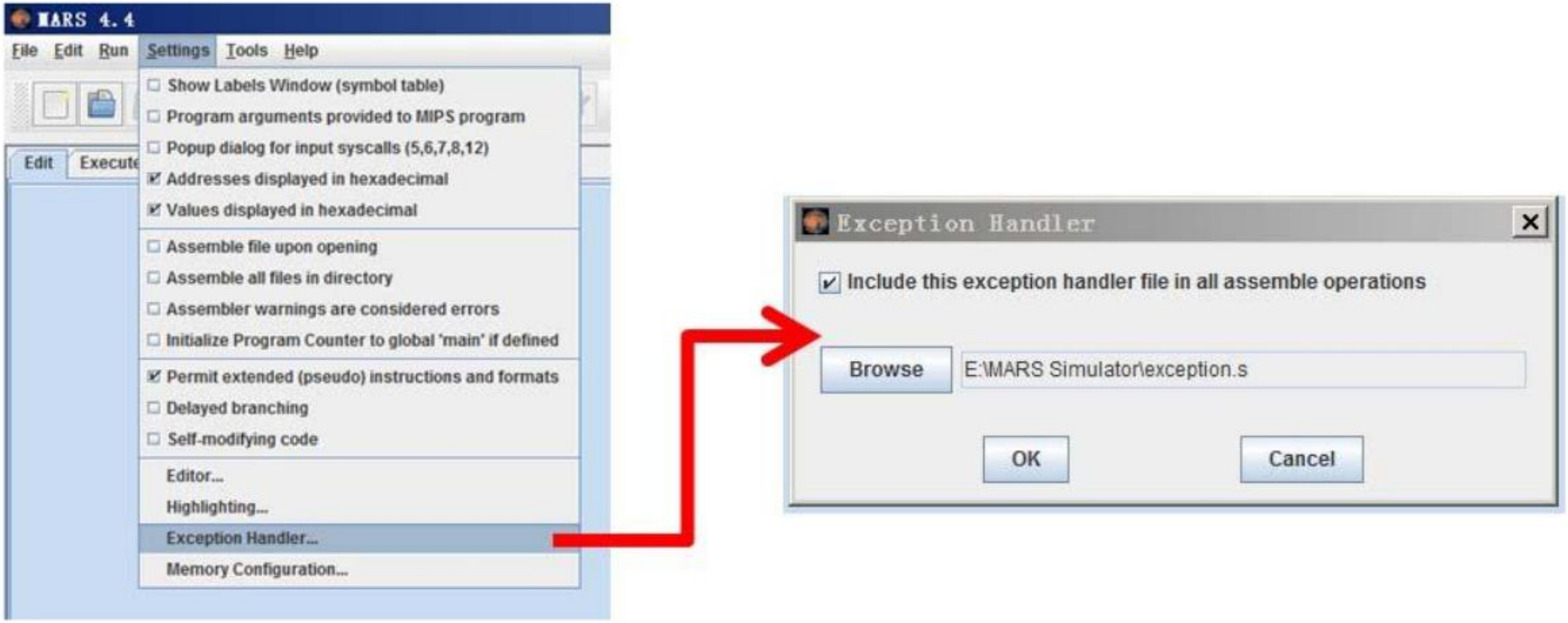
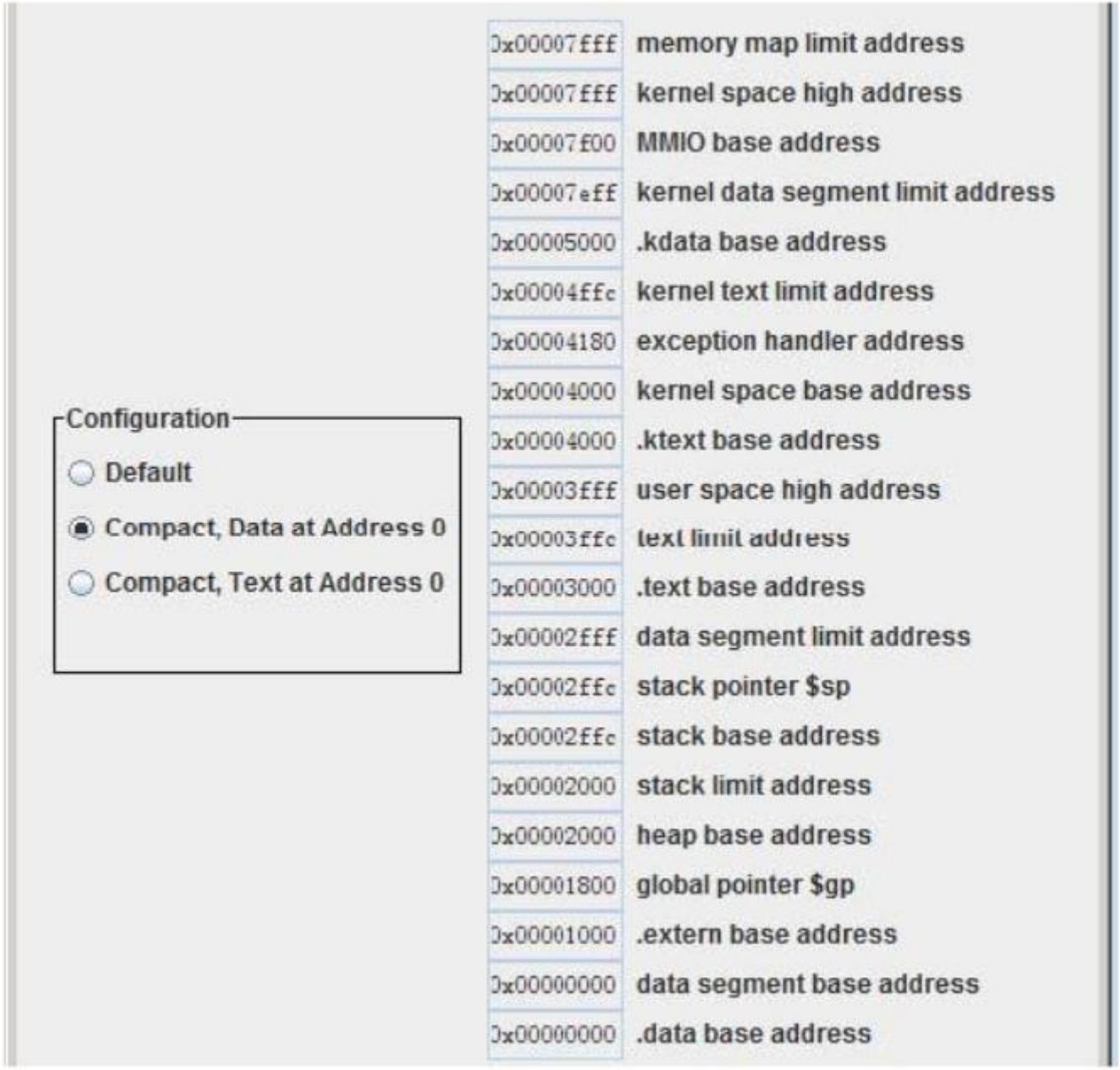


图2 在MARS中验证Exception handler机制

另外，由下图 MARS 的内存分配图我们可以看到，MARS 中的异常入口正是在 0x00004180 处：


Configuration
Default
Compact, Data at Address 0
Compact, Text at Address 0
0x00007fff memory map limit address
0x00007fff kernel space high address
0x00007f00 MMIO base address
0x00007eff kernel data segment limit address
0x00005000 .kdata base address
0x00004ffc kernel text limit address
0x00004180 exception handler address
0x00004000 kernel space base address
0x00004000 .ktext base address
0x00003fff user space high address
0x00003ffc text limit address
0x00003000 .text base address
0x00002fff data segment limit address
0x00002ffc stack pointer $sp
0x00002ffc stack base address
0x00002000 stack limit address
0x00002000 heap base address
0x00001800 global pointer $gp
0x00001000 .extern base address
0x00000000 data segment base address
0x00000000 .data base address

因此，若你的中断处理程序为 exception.s， 样例如下：（注意：该程序中只包含一条 `eret` 指令，直接返回了正常状态。）

```
.ktext 0x00004180
eret
```

思考题

1、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。）

2、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？

十一、CP0 协处理器

CP0 模块在 P7 实验中的主要作用是实现对异常、中断的控制，具体包括 SR、Cause、EPC、PRId 四个寄存器及与中断、异常相关的处理逻辑。

十二、与中断异常相关的四个寄存器

SR 寄存器

- IM[7:2]，即 SR[15:10]，为 6 位中断屏蔽位，分别对应 6 个外部中断。

  相应位置 1 表示允许中断，置 0 表示禁止中断。

- IE，即 SR[0]，为全局中断使能。

  该位置 1 表示允许中断，置 0 表示禁止中断。

- EXL，即 SR[1]，为异常级。

  该位置 1 表示已进入异常，不再允许中断，置 0 表示允许中断。

Cause 寄存器

- IP[7:2]，即 Cause[15:10]，为 6 位待决的中断位，分别对应 6 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]，即 Cause[6:2]，异常编码，记录当前发生的是什么异常。
- BD，即 Cause[31]，记录发生中断的指令是否是延迟槽指令。当 BD 位被置高时，就意味着 EPC 中存储的是发生中断异常的指令的上一指令的地址。

EPC 寄存器

EPC 寄存器负责保存中断/异常时的 PC 值。

PRId 寄存器

通常存入处理器 ID，可以用于实现个性的编码 :)

同学们也可以阅读参考资料获得对这四个寄存器更详细的描述。

十三、CP0 在流水线中的作用

CP0 在流水线中负责响应中断与读写相关寄存器。

CPU 在执行指令时，指令可能会在流水级的不同部位产生异常，如在 E 级可能发生算术指令异常、在 D 级可能发生跳转指令地址异常等。CPU 并不会直接处理这些异常，而是得到一个异常码 `ExcCode`（异常码表示异常的种类，编码的具体规则将在 P7 提交要求章节讲解），并将 `ExcCode` 继续向后流水直至将异常码提交给 CP0。对中断的处理同理，CPU 会将得到的外部中断信号向后流水，直至提交到 CP0。也就是说，CP0 模块负责接收 CPU 产生和接收到的各种中断请求。

CP0 在接收到中断请求后，首先会对请求的合法性进行检验。对于异常产生的中断请求，CP0 在确定当前未响应中断后，响应中断请求；对于接收到的外设中断请求，CP0 在确定 SR 寄存器相应位无中断屏蔽、当前无中断且全局中断使能允许中断后，响应该中断请求。其他情况下，CP0 不会响应中断请求。

CP0 确定响应中断请求意味着 CPU 将跳转至异常处理地址 `0x00004180`，CP0 会向 NPC 模块传递一个跳转信号。在 CPU 进入异常处理程序前，CP0 需要记录当前已进入中断（将 SR 寄存器的 EXL 置位），记录当前的异常码并将指令当前 PC 存储在 EPC 寄存器中（注意，若当前执行的指令为延迟槽指令，则需要存储 PC - 4）。CPU 执行 `eret` 指令从异常返回时，CP0 模块需负责记录当前已结束中断（将 SR 寄存器的 EXL 置 0）。

对于异常产生的中断请求，该存入 EPC 寄存器中的 PC 值是清晰的；但对于外部中断，我们该向 EPC 寄存器中存入哪个 PC 值呢？为此，我们统一引入**宏观 PC** 的概念。

概念解释 **宏观 PC**

宏观 PC 表示整个 CPU “宏观”运行指令所对应的 PC 地址。所谓“宏观”指令，表示该指令之前的所有指令序列对 CPU 的更新已完成，该指令及其之后的指令序列对 CPU 的更新未完成。(宏观 PC 是否字对齐不影响评测)。 更通俗地讲，对于宏观 PC 的意义实际上是指以某一个流水级 PC 作为界限，作为输出端口输出出来，这个流水级一般是你中断信号传入(从 bridge 接入)的流水级，例如：

- 如果你的中断信号在 M 级传入，并且你的中断在 M 级响应，则宏观 PC 为 M 级的 PC （M 级不是气泡的情况下）。
- 如果你的中断信号在 F 级传入，并且随着流水级走到 M 级时才响应，则宏观 PC 为 F 级的 PC。

除了响应中断，CP0 模块承担相关寄存器的读写工作，我们需要在此实现 `mtc0` 和 `mfc0` 两条指令对相关寄存器的读写。

十四、 模块接口

我们给出 CP0 设计的参考接口，同学们也可以根据自己的思考实现。

信号名	方向	用途	产生来源及机制
A1 [4:0]	I	读 CP0 寄存器编号	执行 `mfc0` 指令时产生
A2 [4:0]	I	写 CP0 寄存器编号	执行 `mtc0` 指令时产生
DIn [31:0]	I	CP0 寄存器的写入数据	执行 `mtc0` 指令时产生
PC [31:0]	I	中断/异常时的 PC	
ExcCode[6:2]	I	中断/异常的类型	异常功能部件
HWInt[5:0]	I	6 个设备中断	外部设备
WE	I	CP0 寄存器写使能	执行 `mtc0` 指令时产生

信号名	方向	用途	产生来源及机制
EXLClr	I	置 0 SR 的 EXL 位	执行 `eret` 指令时产生
clk	I	时钟信号	
rst	I	复位信号	
IntReq	O	中断请求	由 CP0 模块确认响应中断
EPC[31:2]	O	EXC 寄存器输出至 NPC	
DOut[31:0]	O	CP0 寄存器的输出数据	执行 `mfc0` 指令时产生

十五、 参考资料

CP0 设计及其相关指令的实现，以及硬软件在中断处理上的协同是 P7 中最有挑战性的部分。仅阅读教程中的简要介绍远远不够，因此课程组放出一些推荐阅读的资料，希望同学能加以研究，尝试去理解其中的思路。

届时我们也会在讨论区放出官方答疑贴，但我们仅会对我们认为有必要回答的问题进行回答（涉及测试的统一要求）。

推荐资料列表：

12. L13-MIPS 系统结构-V1.pdf
13. 《See MIPS Run Linux》中相关章节
14. 《计算机组成与设计：硬件/软件接口》中相关章节
15. Google / Bing 等搜索引擎
16. 讨论区

十六、P7 提交要求

在 P7，你将在 P6 的基础上进一步完善，通过加入 CP0、Bridge、计时器等硬件，修改现有 CPU 结构，支持教程所指定的几类异常与中断，形成简易的 MIPS 微系统。请注意，本节所讲内容需要同学们已经掌握了中断与异常的理论知识，并且本章仅对必要内容予以说明，因为在 P7 阶段我们希望同学们能够自主阅读有关材料，并在严密的逻辑下自主设计出硬件，所以“一千位同学就有一千个不同的 P7 设计”。

相信你一定能够能完成 :)

概念解释

本章节中，”中断“和”异常“两词将分别指代，不再如上一章节将异常视作内部的中断。

十七、设计规格说明

1. MIPS 微系统整体要求：
    1. MIPS 处理器须为流水线设计，MIPS 微系统须支持中断和异常。
    2. 除本文明确的规范和补充声明外，MIPS 微系统设计以《See MIPS Run Linux》(下文简称《SMRL》)作为标准。《SMRL》的标准与 MARS 的行为存在一定差异，在测试时不以 MARS 为准。
    3. P7 相较于前几个 Project 为同学预留了更多需要自主设计的内容，最终 P7 的实现因人而异。只要设计满足所给出的设计约束、行为规范和 mips 基本设计规范，任何设计都被认为是正确的。
2. MIPS 微系统接口(请顶层模块严格满足该要求)

```
module mips(
    input clk,                  // 时钟信号
```

```
    input reset,                    // 同步复位信号
    input interrupt,                // 外部中断信号
    output [31:0] macroscopic_pc,   // 宏观 PC（见下文）

    output [31:0] i_inst_addr,      // 取指 PC
    input  [31:0] i_inst_rdata,     // i_inst_addr 对应的 32 位指令

    output [31:0] m_data_addr,      // 数据存储器待写入地址
    input  [31:0] m_data_rdata,     // m_data_addr 对应的 32 位数据
    output [31:0] m_data_wdata,     // 数据存储器待写入数据
    output [3 :0] m_data_byteen,    // 字节使能信号

    output [31:0] m_inst_addr,      // M 级 PC

    output w_grf_we,                // grf 写使能信号
    output [4 :0] w_grf_addr,       // grf 待写入寄存器编号
    output [31:0] w_grf_wdata,      // grf 待写入数据

    output [31:0] w_inst_addr       // W 级 PC
);
```

3. P7 相对于 P6 的顶层模块新增了两个接口：
    1. macroscopic_pc[31:0]：宏观 PC，相应概念详见 CP0 协处理器章节。我们保证，在评测过程中 macroscopic_pc[31:0] 仅用于寻找 CPU 可能的“薄弱”状态并以此为依据产生中断信号。
    2. interrupt：来自系统外的中断信号，每次中断信号会持续到处理器向 0x7F20 写入值为止(可以参考我们给出的 tb 理解)，处理该中断信号的方式与处理内部 timer 模拟的中断信号行为应保持一致。
4. MIPS 处理器应该支持 MIPS-C4 指令集，在 P6 基础上新增了{eret, mfc0, mtc0}三条新指令。

指令序号	指令符号	指令说明
1	lb	取有符号字节
2	lbu	取无符号字节
3	lh	取有符号半字
4	lhu	取无符号半字
5	lw	取全字
6	sb	存字节
7	sh	存半字
8	sw	存全字
9	add	加
10	addu	加(不检测溢出)
11	sub	减
12	subu	减(不检测溢出)
13	mult	有符号乘
14	multu	无符号乘
15	div	有符号除
16	divu	无符号除

指令序号	指令符号	指令说明
17	sll	逻辑左移立即数
18	srl	逻辑右移立即数
19	sra	算术右移立即数
20	sllv	逻辑左移
21	srlv	逻辑右移
22	srav	算术右移
23	and	按位与
24	or	按位或
25	xor	按位异或
26	nor	按位或非
27	addi	加立即数
28	addiu	加立即数(不检测溢出)
29	andi	按位与立即数
30	ori	按位或立即数
31	xori	按位异或立即数
32	lui	加载立即数至高位

指令序号	指令符号	指令说明
33	slt	小于则置位(有符号)
34	slti	小于立即数则置位(有符号)
35	sltiu	小于立即数则置位(无符号)
36	sltu	小于则置位(无符号)
37	beq	相等则转移
38	bne	不等则转移
39	blez	小于等于 0 则转移
40	bgtz	大于 0 则转移
41	bltz	小于 0 则转移
42	bgez	大于等于 0 则转移
43	j	无条件跳转至立即数地址
44	jal	无条件跳转至立即数地址并链接
45	jalr	无条件跳转至寄存器地址并链接
46	jr	无条件跳转至寄存器地址
47	mfhi	从 HI 取值
48	mflo	从 LO 取值

指令序号	指令符号	指令说明
49	mthi	向 HI 存值
50	mtlo	向 LO 存值
51	mfc0	从 CP0 取值
52	mtc0	向 CP0 存值
53	eret	恢复用户态

Note

测试数据中可能出现 eret 指令后紧跟另一条非 nop 指令的情况，你的设计应该保证 eret 的后续指令不被执行。

5. 本次 MIPS 微系统需要支持的异常：

ExcCode	助记符	描述
0	Int	中断
4	AdEL	取数或取指时地址错误
5	AdES	存数时地址错误
10	RI	不认识的(或者非法的)指令码
12	Ov	自陷形式的整数算术指令(例如 add)导致的溢出

十八、 MIPS 微系统设计约束

6. 顶层模块设计约束
    1. 顶层模块中应该至少包含 CPU、Bridge、Timer0、Timer1 四个功能部件。
        - CPU：在 P6 基础上添加相应扩展。
        - Bridge：须作为独立的 module，不包括在 CPU 中。
        - Timer0、Timer1：计时器。
    2. Timer0 输出的中断请求信号接入 HWInt[0] (最低中断位)，Timer1 输出的中断请求信号接入 HWInt[1]，来自 MIPS 微系统外部的中断请求信号接入 HWInt[2]。
7. CPU 功能部件设计约束
    1. IM：容量为 20KB (32bit/word x 5Kword)。
    2. DM：容量为 12KB (32bit/word x 3Kword)。
8. 地址空间

	地址或地址范围	**备注**
数据存储器	0x0000_0000 至 0x0000_2FFF	
指令存储器	0x0000_3000 至 0x0000_6FFF	
PC 初始值	0x0000_3000	
Exception Handler 入口地址	0x0000_4180	
计时器 0 寄存器地址	0x0000_7F00 至 0x0000_7F0B	计时器 0 的 3 个寄存器
计时器 1 寄存器地址	0x0000_7F10 至 0x0000_7F1B	计时器 1 的 3 个寄存器

十九、 CP0 协处理器设计约束

9. 协处理器位置：不作明确要求，需自行设计。
10. 输出要求：写入时不需要 display。
11. 寄存器：为了支持异常和中断,流水线必须支持 0 号协处理器(CP0)，必须实现的寄存器包括:**SR、CAUSE、EPC、PrID**。

CP0 寄存器名称	读写	寄存器规范
SR(状态寄存器)	R/W	IM7-2(设备中断使能)， EXL(是否处于内核态)， IE(全局中断使能)。
CAUSE(原因寄存器)	R	IP7-2(设备中断请求)，EXC(异常码)，BD(分支延迟)，其余位数请保持 0，测试程序不会向未实现位写值。
EPC(异常 PC 寄存器)	R/W	EPC(异常返回点)，异常时存入值参考《SMRL》，中断时请自行设计。中断异常发生时 EPC 写入值不需要字对齐。
PR(处理器寄存器)	R	0xDEAD_C0DE(自定义型号)。

12. SR 寄存器初始值可以自行设计，在测试中断时软件会在一开始为 SR 寄存器赋初值。SR 寄存器未实现位读出保持 0 即可。

二十、 计时器设计约束

13. 定时器官方源代码已经给出，无需自行设计实现，但需要按照前文教程要求提交定时器的相关说明文档。
14. 定时器的寄存器被写入时不用 display。

二十一、概念解释及实现说明

15. 受害指令：受害指令是与异常相关的概念，对中断是否指定受害指令不做要求。当受害指令提交至 CP0 时，便会触发对应异常。这里我们需要将”受害“和”犯罪“的概念相区分：对于多数异常的情况“受害=犯罪”；但少数情况下，“受害 != 犯罪”，例如对于 j、jr、jal、jalr 指令，若跳转至不对齐的地址，则受害指令应当是 PC 值不正确的指令（即需要向 EPC 写入不对齐的地址）。
16. 执行效果：我们可以这样简单地看：硬件 = 组合逻辑+时序逻辑，真正有“记忆”的是时序逻辑，因而一条或多条指令的执行效果可以被理解为对内存、各类寄存器等时序部件产生的影响。有些指令连续地重复执行多次后时序部件数据可能保持一致，如 mul、mthi 等；而有的则并不一定保持一致，如 add、sub 等。
17. 嵌套中断异常：本实验不要求支持中断异常嵌套情况。
18. 优先级：中断优先级高于异常优先级，即当有异常提交至 CP0 寄存器时，若有中断发生，则硬件应先响应中断，并重新执行受害指令及其后续指令；若没有中断发生，则处理异常。
19. CP0 寄存器：当中断异常发生时，会向 CP0 寄存器对应位写入相关值，至少需要实现：屏蔽所有中断、记录中断异常类型、记录是否分支延迟(BD)、在 EPC 记录地址。
20. 精确异常：对所有异常都应遵循精确异常的处理规则，即受害指令的前序指令的期望的执行效果都应执行，并且受害指令的后序指令执行效果不影响异常处理程序返回后所执行指令的执行效果。通俗来讲，就是流水线中的每条指令要么执行完毕，要么根本就不产生任何效果。(此部分需要结合软件的行为规范理解)
21. 中断规范：本实验不要求在中断发生时指定具体受害指令，但需要保证中断的处理是精确的，进入处理程序前后每条指令的执行效果不变。(此部分需要结合软件的行为规范理解)
22. 细节规范：

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

异常与中断码	助记符与名称	指令与指令类型	描述
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，lhu	取数地址未与 2 字节对齐		
lh，lhu，lb，lbu	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 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 的范围		

异常与中断码	助记符与名称	指令与指令类型	描述
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

23. 补充说明：

- 分支跳转指令无论跳转与否，延迟槽指令为受害指令时 BD 均需要置位。
- 发生取指异常后视为 nop 直至提交到 CP0。
- 发生 RI 异常后视为 nop 直至提交到 CP0。
- load 与 store 类算址溢出按照 AdEL 与 AdES 处理。
- 测试时不会出现跳转到未加载指令的位置。
- 在 P7 的测试中，对于未知指令的判断仅需考虑 Opcode(和 R 型指令的 funct)，未知指令的测试点中，非法指令的 Opcode 和 Func 码组合一定没有在正确指令集中出现。

二十二、 软件行为规范

软件行为是指中断响应程序中指令序列执行所产生的功能。本节所描述的软件行为规范旨在帮助同学们加深对整个中断异常流程的理解，无需实现该部分内容。

24. 异常与中断的处理流程：

Step1-引导与识别：读取 Cause 和 EPC 寄存器，判断错误类型。

Step2-构造异常处理环境：保存现场。

Step3-处理异常：根据异常类型和其他属性执行对应处理。

Step4-准备返回：恢复现场。

Step5-从异常返回：eret 指令。

针对各类中断和异常，软件对应 Step3 具体对应的处理是：

- 若为中断，则视情况对外设进行操作(如修改计时器的设置)，EPC 不作修改。
- 若为取数异常/存数异常/算术溢出，则视情况修改 EPC 为受害指令的下一条。
- 若为取指异常/RI，则更改 EPC 的值为受害指令的下一条。

25. 定时器操作规范
    - 软件只会对定时器进行一些正常并且合理的操作，对于定时器的操作一共有两部分：一部分是程序设置两个定时器的模式；另一部分是中断发生进入异常处理后，软件会修改定时器的一些设置，并且先停止计数，修改完其它设置后再允许计数。除这两种情况外，测试程序保证 CPU 正确执行时不会再对定时器的设置进行修改。
26. 官方测试数据说明
    - eret 只会出现在异常处理程序中。
    - 软件保证不会写入 Cause 和 PRId，但可能写入 EPC。
    - 中断处理程序除了上述描述的操作外，还会进行一些对寄存器和内存的读写来验证 CPU 的正确性。
    - 中断处理程序中保证不出现异常。
    - 测试程序只会在一开始设置 EXL 为 0，设置 IE 为 1，接下来不再会对 EXL 和 IE 进行设置。

二十三、 官方测试说明

27. 外设不给予中断时，使用的 tb 为：下载链接。
28. 评测机通过检测同学们的宏观 PC 给予中断信号并对中断进行测试，例如此下载链接中的 tb 会在处理器的宏观 PC 第一次到达 0x3010 时给予 CPU 一个中断信号。
29. 写 0x7F20 的行为由 CPU 硬件完成，而非由软件完成，测试指令序列中出现写 0x7F20 的行为应触发相应的异常，为帮助同学们更好地理解，课程组提供了一个响应外部中断的示例：下载链接。

30. 为便于进行测试，我们允许从 0x417C 直接前进到 0x4180，此种情况下 CPU 行为与 P6 一致，不应有中断响应等其他行为。

二十四、 补充声明

31. 异常入口：《SMRL》的表 5.1 中定义了 MIPS 的异常入口，但考虑到简化设计以及与 MARS 模拟器保持一致，我们将只支持 0x4180 一个入口地址，所有异常与中断都将从这里进入。
32. EXL：当进入中断或异常状态时，均需要将 EXL 置为 1，用以屏蔽中断信号，《SMRL》中并没有指定进入中断时 EXL 的值。
33. Cause 寄存器的 IP 域每周期写入 HWint 对应位的值即可。

二十五、 更正记录

34. 2021.12.12：保证测试程序不会在用户态（可简单理解为未陷入中断异常时）将 EXL 置为 1。

思考题汇总

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？)

2、BE 部件对所有的外设都是必要的吗？

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。

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。)

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？