32位MIPS处理器实验设计文档

Cache小组

2015年1月9日

目录

1 引言

1.1 编写目的

在此前编写的需求文档中，已经明确了此次联合实验预期达到的目标，实验中需要完成的各部分工作，也对实验中需要用到的关键技术做了简要的原理性说明，此次实验的前期准备工作的需求文档中基本体现。

进入实际的代码开发阶段，VHDL代码的编写需要更加详细的接口，更加精准的功能说明，更加细化的流程控制。从前的需求文档已经不足以对开发过程进行具体的指导了，需要一份更加详细的设计文档。

因此，为了指导代码的实际开发过程，编写此设计文档。

设计文档预期读者为任务提出者：刘卫东老师、李山山老师、白晓颖老师。未来需要完成此实验的同学也可参考本文档进行设计。

1.2 背景

系统名称：32位MIPS处理器
任务提出者：计算机组成原理课程：刘卫东老师、李山山老师
　　软件工程课程：白晓颖老师
开发者：计23 李天润
　　计23 胡津铭
　　计23 孙皓

1.3 参考资料

实验指导文档
OsLab实验参考文档
计算机组成原理综合实验报告 贾开
《计算机组成和设计 硬件/软件接口》
《See MIPS Run》

2 模块设计

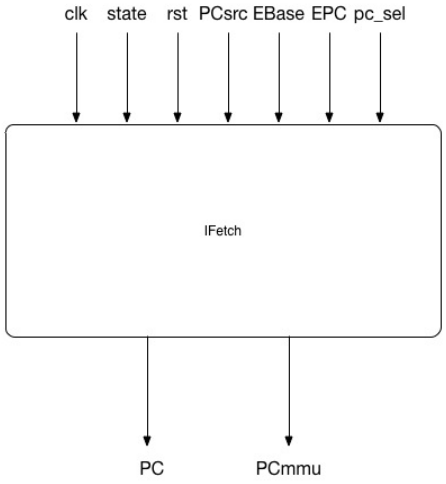
2.1 取指令模块

2.1.1 端口说明

端口名	端口方向	端口类型
	端口描述	
clk	in	std_logic
	CPU时钟信号	
rst	in	std_logic
	初始化信号，在CPU启动时使用。	
state	in	status（自定义状态集合）
	CPU当前状态	
mmu_ready	in	std_logic
	标志访存是否结束	
PCSrc	in	std_logic_vector(31 downto 0)
	说明：非异常状态下的指令地址。 来源：WB模块。 到达时间：当前指令InsF上升沿之前。 产生时间：上一条指令WB上升沿之后。	
EBase	in	std_logic_vector(31 downto 0)
	说明：异常处理基地址。 来源：CP0模块。 到达时间：当前指令InsF上升沿到来之前。	
EPC	in	std_logic_vector(31 downto 0)
	说明：ERET指令的返回地址。 来源：CP0模块。 到达时间：当前指令InsF上升沿到来之前。	
pc_sel	in	std_logic_vector(1 downto 0)
	pc_sel(1): 说明：eret_enable，使能信号。 来源：ID模块。 产生时间：上一条指令ID上升沿之后。 持续时间：直到下一条指令的IF阶段。	

	pc_sel(0)： 说明：pc_control，判断是否为异常状态。 来源：异常模块。 到达时间：当前指令InsF上升沿到来之前。	
PC	out	std_logic_vector(31 downto 0)
	说明：PC寄存器，时序逻辑。 产生时间：当前指令InsF上升沿到来之后。 有效时间：直到下一条指令的IF阶段。	
PCmmu	out	std_logic_vector(31 downto 0)
	说明：为MMU单元提供的PC，组合逻辑。 产生时间：当前指令InsF上升沿到来之前。 有效时间：直到当前指令IF阶段结束。	

图 1: 取指令模块端口示意图



2.1.2 内部实现

需要的数据有PcSrc、EBase、EPC。

PcSrc产生于上一条指令的WB阶段。EBase为固定值，直接从CP0部分连接过来。EPC在异常阶段写入，直接从CP0部分连接过来。pc_sel为上一条指令的解码阶段产生。因此，所有的数据都能够在InsF时钟上升沿之前准备完毕。

内部分为两个部分对PC进行计算：首先为组合逻辑部分，需要在InsF时钟上升沿到来之前为MMU计算出PC值，通过条件赋值语句，即时计算出PC值，连接到MMU部分，使得MMU能够在InsF上升沿进行取指令的操作。

其次为时序逻辑部分，在InsF时钟上升沿时，对PC进行选择，选择方式与PCmmu相同。且由于访存可能持续多个时钟周期，因此需要对mmu_ready位进行判断，保证只在访存的第一个时钟周期锁存PC的值。此process产生的PC，在当前指令的全部周期有效，是计算RPC、branch、jump的地址的基础。

2.2 指令解析模块

2.2.1 端口说明

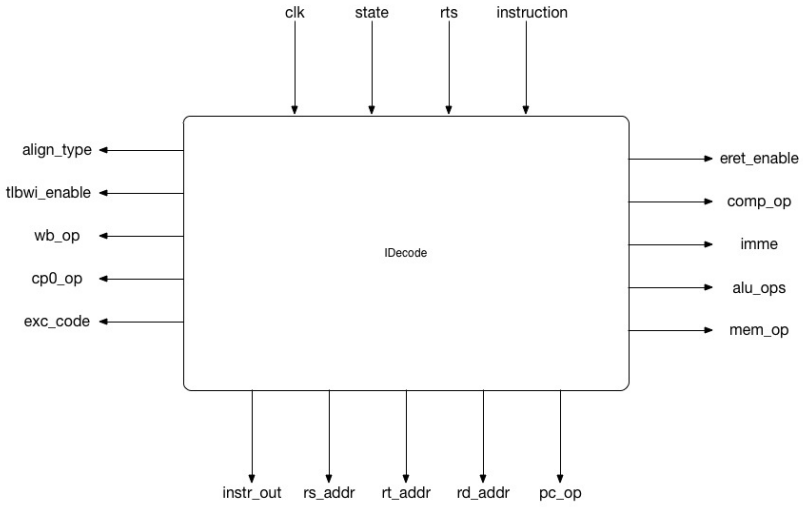
端口名	端口方向	端口类型
	端口描述	
clk	in	std_logic
	CPU时钟信号。	
rst	in	std_logic
	初始化信号，在CPU启动时使用。	
state	in	status（自定义状态集合）
	CPU当前状态。	
instruction	in	std_logic_vector(31 downto 0)
	说明：当前指令。 来源：MMU模块。 到达时间：当前指令InsD上升沿之前。 产生时间：当前指令InsF上升沿之后。	
instr_out	out	std_logic_vector(31 downto 0)
	说明：指令寄存器，除三个寄存器的编号，其他所有控制线均从此产生。之后周期中如果需要用到指令也从此处获得。 产生时间：当前指令InsD上升沿之后。	
rs_addr	out	std_logic_vector(4 downto 0)
	说明：通用寄存器编号1，在InsD阶段需要读取到值。 产生时间：当前指令InsD上升沿之前。	
rt_addr	out	std_logic_vector(4 downto 0)
	说明：通用寄存器编号2、写入寄存器编号，在InsD阶段需要读取到值。 产生时间：当前指令InsD上升沿之前。	
rd_addr	out	std_logic_vector(4 downto 0)
	说明：CP0寄存器编号、写入寄存器编号，在InsD阶段需要读取到值。 产生时间：当前指令InsD上升沿之前。	

pc_op	out	std_logic_vector(1 downto 0)
	说明：PCSrc选择器，正常状态下PC的选择方式。输出到WB模块，4选1数据选择器的控制信号，选择正确的PC。 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsD上升沿之前。 00 PC+4的计算结果。 01 branch指令PC计算的结果。 10 jump指令目标地址。 11 主ALUOut计算结果。	
eret_enable	out	std_logic
	说明：ERET使能，对PC进行选择。专门对ERET指令使用，输出到IFetch模块。 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsF上升沿之前。	
comp_op	out	std_logic_vector(2 downto 0)
	说明：比较信号，branch指令的跳转条件。输出到WB模块，如果为branch系列指令则通过此信号进行选择。 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsD上升沿之前。 000 BEQ（R[s] = R[t]） 001 BGEZ(R[s] >= 0) 010 BGTZ(R[s] > 0) 011 BLEZ(R[s] <= 0) 100 BLTZ(R[s] < 0) 101 BNE(R[s] != R[t])	
imme	out	std_logic_vector(31 downto 0)
	说明：32位立即数，针对不同指令的需求产生。立即数本身作为ALUSrc的来源之一。 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsD上升沿之前。 立即数产生方式有四种，均为从不同立即数长度扩展为32位。 16位有符号扩展，16位无符号扩展，移位指令需要的5位到32位扩展，jump指令的立即数扩展。	
alu_ops	out	std_logic_vector(8 downto 0)
	说明：控制ALU模块。 ALUSrcA(8 downto 7)：ALU第一输入的选择信号，四选一数据选择。 ALUSrcB(6 downto 5)：ALU第二输入的选择信号，四选一数据选择。	

	ALUOp(4 downto 0)：ALU操作，5位，整合了乘法器相关运算，详细说明见ALU模块说明。 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsD上升沿之前。	
mem_op	out	std_logic_vector(2 downto 0)
	说明：控制MEM模块。 MEMRead(2)：是否可读内存。 MEMWrite(1)：是否可写内存。 MEMValue(0)：选择写入内存的值，寄存器的数据或者SB指令处理之后的数据。 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsD上升沿之前。	
wb_op	out	std_logic_vector(4 downto 0)
	说明：控制WB模块。 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsD上升沿之前。 RegDst(5 downto 4)：写回寄存器编号。 如果为00则写回16到20位rt寄存器，如果为01则写回11到15位rd寄存器，如果为10则写回31号寄存器。 如果为11则不写寄存器。 RegValue(3 downto 1)：写回寄存器的内容 000 主ALU计算结果 001 读取内存的数据 010 RPC 011 Zero-extend内存byte 100 Signed-extend内存byte 101 Zero-extend内存word 110 CP0寄存器	
cp0_op	out	std_logic_vector(1 downto 0)
	说明：控制CP0模块。 EPCValue(1)：异常产生时，EPC写入的内容，选择写入PC或者PC+4。 CP0Write(0)：CP0寄存器是否可写。 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsD上升沿之前。	
tlbwi_enable	out	std_logic
	说明：TLB写使能。 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsD上升沿之前。	
align_type	out	std_logic_vector(1 downto 0)

	说明：访存对齐方式 产生时间：当前指令InsD上升沿之后。 有效时间：下一条指令InsD上升沿之前。	
exc_code	out	std_logic_vector(1 downto 0)
	说明：指令解析模块产生的异常 产生时间：当前指令InsD下降沿之后，在下一条指令的下降沿时消除 有效时间：一个CPU时钟周期，需要异常模块及时处理。 00 没有产生异常 01 syscall 10 未定义的指令异常	

图 2: 指令解析模块端口示意图



2.2.2 内部实现

需要的数据有instruction，产生于当前指令InsF上升沿之后，能够在当前指令InsD上升沿之前到达。

内部实现有一下四个要点：

1. 在InsD阶段除了解码指令之外，还需要读取通用寄存器和CP0寄存器的值。需要在InsD上升沿到来之前，将三个寄存器编号发送给寄存器堆。该功能在解码模块之外的顶层模块实现，解码模块中输出的rs,rt,rd三个寄存器编号，有效期均为整条指令。

2. 其他控制线的生成均通过时钟驱动，在IDEcode有相应寄存器，连接至输出端。在InsD时钟上升沿之后，根据指令解码产生控制信号。产生的控制信号根据需要，输出到ALU、WB、MEM、CP0、IFetch、MMU模块。

3. 除eret_enable信号之外，所有时序逻辑产生的信号有效期均到下一条指令的解码阶段。在lab5中出现syscall异常返回后立刻在取指阶段触发tlbmiss异常，因此需要把eret_enable信号在取指阶段消除，否则触发tlbmiss时并没有进入解码阶段，eret_enable信号就不会被消掉。

4. 异常：

   在指令解码阶段可能产生两种异常：系统调用或者未定义的指令异常。由于ALUOp的生成需要使用case语句对每条指令单独处理，因此选择在case的when others部分生成未定义的指令异常如果是syscall则产生异常，并且将ALUOp置为无用的操作。

   异常信号产生的时间为指令解码阶段的时钟下降沿，直接通过exc_code输出到异常处理模块，再利用其控制CPU主状态机的工作流程。

   异常在下一个时钟的下降沿会被消除，因此需要异常处理模块即使将异常信息捕获。

2.3 ALU模块

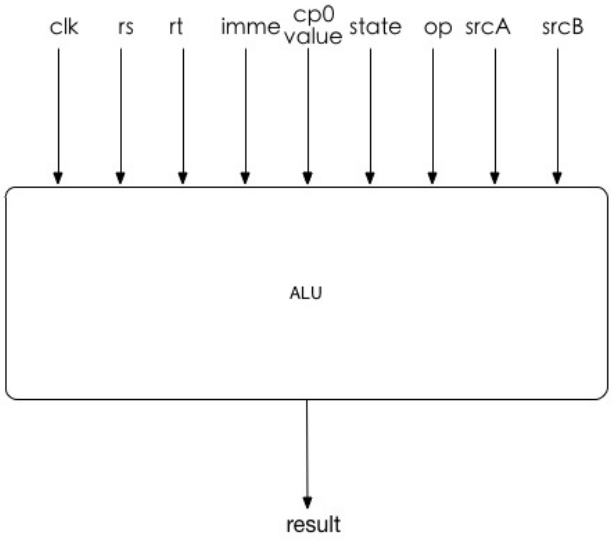
2.3.1 端口说明

端口名	端口方向	端口类型
	端口描述	
clk	in	std_logic
	CPU时钟信号。	
rs_value	in	std_logic_vector(31 downto 0)
	来自通用寄存器堆的第一个值。需要在clk时钟上升沿之前准备好，并保持到clk时钟上升沿之后一极短时间。	
rt_value	in	std_logic_vector(31 downto 0)
	来自通用寄存器堆的第二个值。需要在clk时钟上升沿之前准备好，并保持到clk时钟上升沿之后一极短时间。	
imme	in	std_logic_vector(31 downto 0)
	指令中包含的立即数，来自指令解析模块。需要在clk时钟上升沿之前准备好，并保持到clk时钟上升沿之后一极短时间。	
cp0_value	in	std_logic_vector(31 downto 0)
	来自cp0寄存器，mfc0指令需要。需要在clk时钟上升沿之前准备好，并保持到clk时钟上升沿之后一极短时间。	
state	in	status（自定义状态集合）

	来自状态控制模块，用来指示当前处于工作状态的模块。若当前非ALU工作状态，则任何外部输入都不会对ALU的hi、lo寄存器以及ALU的输出造成修改。需要在clk时钟上升沿之前准备好，并保持到clk时钟上升沿之后一极短时间。	
alu_op	in	std_logic_vector(4 downto 0)
	来自指令解析模块的ALU运算操作符。需要在clk时钟上升沿之前准备好，并保持到clk时钟上升沿之后一极短时间。各操作符代表的意义为（A、B分别代表经过alu_srcA、alu_srcB选择后的值，result代表alu_result，lo、hi代表乘法寄存器）： 00000 result = A + B 00001 result = A - B 00010 result = A - B（比较大小，实际做减法） 00011 result = A & B 00100 result = A | B 00101 result = A ^ B 00110 result = ~(A | B) 00111 result = B << A 01000 result = B >> A（算术右移） 01001 result = B >> A（逻辑右移） 01010 result = A < B?（有符号比较，结果真时最低位输出1，否则输出0，其他位总是输出0） 01011 result = A < B?（无符号比较，结果真时最低位输出1，否则输出0，其他位总是输出0） 10000 hi_lo = A * B（补码乘法） 10001 result = lo 10010 result = hi 10011 lo = A 10100 hi = A	
alu_srcA	in	std_logic_vector(1 downto 0)
	ALU的第一个操作数的选择码，当值为“00”时选取rs_value，当值为“01”时选取imme，当值为“10”时选取cp0_value，当值为“11”时选取立即数16。需要在clk时钟上升沿之前准备好，并保持到clk时钟上升沿之后一极短时间。	
alu_srcB	in	std_logic_vector(1 downto 0)
	ALU的第二个操作数的选择码，当值为“00”时选取rt_value，当值为“01”时选取imme，当值为“10”时选取cp0_value。需要在clk时钟上升沿之前准备好，并保持到clk时钟上升沿之后一极短时间。	
alu_result	out	std_logic_vector(31 downto 0)

ALU的输出，在下一个时钟上升沿之前准备好，并保持直到下一次能使ALU输出改变（state为ALU工作状态、alu_op非写hi、lo寄存器）的时钟上升沿。

图 3: ALU模块端口示意图



2.3.2 内部实现

每次时钟上升沿到来时，检查state，若不是ALU工作状态则不进行任何操作。根据alu_srcA选择第一个操作数，根据alu_srcB选择第二个操作数，根据操作码进行相应的运算，将结果输出或保存到hi、lo寄存器中。

注意乘法运算需要较多的时间，因此若在某一个时钟上升沿进行乘法运算，不能认为在下一个时钟上升沿就能在hi、lo寄存器中取到正确的结果。一般来说，在25MHz时钟频率下进行乘法运算在1个时钟周期内可以完成，那么如果在此频率运行的CPU上，可以保证两条连续的乘法、取hi(lo)寄存器指令能得到正确的结果。乘法运算时间需要根据不同的硬件平台、时钟频率进行测量，不能一概而论。

2.4 访存模块

2.4.1 端口说明

端口名	端口方向	端口类型

	端口描述	
result	in	std_logic_vector(31 downto 0)
	说明：访存地址 来源：ALU模块 到达时间：指令执行阶段时钟上升沿后。 保持时间：至少保持到访存阶段时钟上升沿之后。	
rst	in	std_logic
	初始化信号，在CPU启动时使用。	
mem_op	in	std_logic_vector(2 downto 0)
	说明：内存操作控制线 来源：IDecode模块 到达时间：指令解码时钟上升沿之后。 保持时间：至少保持到访存阶段结束。 mem_op(2): memRead：内存读使能。 mem_op(1): memWrite：内存写使能。 mem_op(0): memValue：内存写入数据。 如果为0则写入来自寄存器的数据，如果为1则写入SB指令处理之后的数据。	
rt_value	in	std_logic_vector(31 downto 0)
	说明：来自寄存器的数值，访存阶段写入数据的可能来源之一。 来源：通用寄存器。 到达时间：指令解码时钟上升沿之后。	
mmu_value	in	std_logic_vector(31 downto 0)
	说明：访存得到的数据，为SB指令提供支持。 来源：MMU模块。 到达时间：第二次访存操作上升沿之前。 产生时间：第一次访存操作下降沿之后。	
addr_mmu	out	std_logic_vector(31 downto 0)
	说明：访存阶段的地址。 产生时间：访存阶段时钟上升沿之前。	
write_value	out	std_logic_vector(31 downto 0)
	说明：访存阶段写入数据。 产生时间：访存阶段时钟上升沿之前。	
read_enable	out	std_logic
	说明：访存读使能。	
write_enable	out	std_logic
	说明：访存写使能。	

2.4.2 内部实现

1. MEM模块在指令的执行周期之后，但是内存访问模块需要使用访存周期的时钟上升沿，因此MEM模块不能占用时钟上升沿，需要设计为一个组合逻辑模块。其主要功能为访存周期的预处理，为其生成访存的地址与写入的数据，以及产生使能信号。

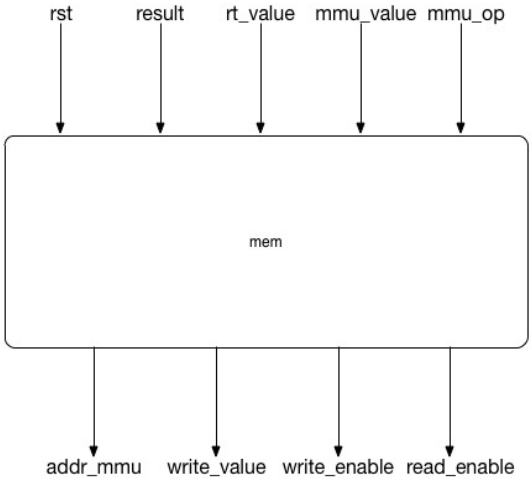
2. MEM模块内部完全由组合逻辑实现。

   目前MEM模块有一些输入输出之间采取的是直接连线的方式，主要目的在于比较清晰地体现出访存的设计思路。后期可能将部分连线直接连接到MMU模块，或者在Xilinx编译与优化的过程中，由编译器直接优化掉。

   直接连线部分：addr_mmu输出直接与输入的result端口相连，将访存地址直接输出到MMU模块。read_enable输出直接与输入的mem_op(2)相连，将读使能输出到MMU模块。write_enable输出直接与输入的mem_op(1)相连，将写使能输出到MMU模块。

   组合逻辑部分：wrirte_value为内存写入值，需要根据memValue信号进行选择。如果memValue为0，则write_value的取值为rt_value，写入寄存器2的值。如果memValue为1，说明此条指令为SB指令，有两个访存阶段。第一访存阶段为读，此时不需要write_value的值。第二访存阶段为写，在此阶段开始之前，第一访存阶段已经取出内存中addr_mmu地址中的数据，在此基础上进行一个byte的修改，利用addr_mmu最后两位选择修改位置，将rt_value的最低位byte写入，整体作为内存的写入值传递给MMU模块。

图 4: 访存模块端口示意图



2.5 写回模块

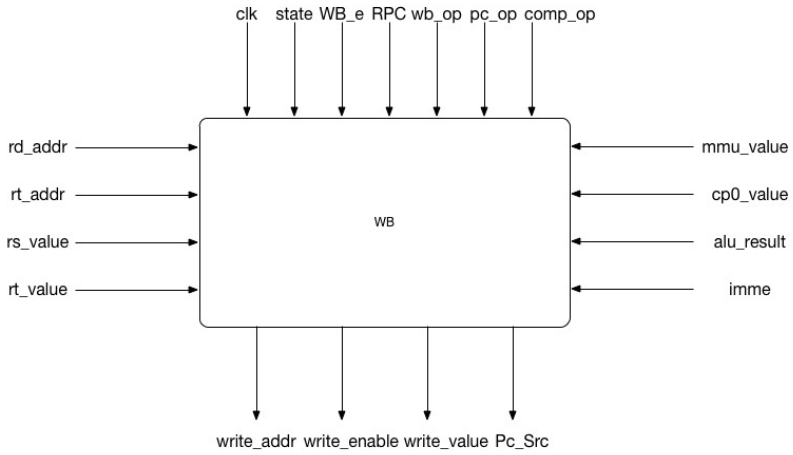
2.5.1 端口说明

端口名	端口方向	端口类型
	端口描述	
clk	in	std_logic
	CPU时钟信号	
state	in	status（自定义状态集合）
	CPU当前状态	
WB_e	in	std_logic
	WB文件的使能信号	
RPC	in	std_logic_vector(31 downto 0)
	即本周期的PC+4，要求ALU阶段上升沿之前准备好。	
mmu_value	in	std_logic_vector(31 downto 0)
	来自MMU的读取值，要求WB阶段上升沿之前准备好。	
cp0_value	in	std_logic_vector(31 downto 0)
	来自CP0的读取值，要求WB阶段上升沿之前准备好。	
alu_result	in	std_logic_vector(31 downto 0)
	来自ALU的读取值，要求WB阶段上升沿之前准备好。	
wb_op	in	std_logic_vector(4 downto 0)
	WB阶段写入寄存器编号的来源与写入数据的来源选择，要求WB阶段上升沿之前准备好。 4-3位为写入寄存器编号来源的控制信号： 00表示写入rt寄存器； 01表示写入rd寄存器； 10表示写入31寄存器； 11或其他信号表示不写； 2-0位为写入数据来源的控制信号： 000表示数据来自ALU； 001表示数据来自MMU； 010表示数据来自RPC； 011表示数据来自MMU，此时为LBU操作，根据alu_result的低2位决定使用MMU数据的哪一字节，高位零扩展；	

	100表示数据来自MMU，此时为LB操作，根据alu_result的低2位决定使用MMU数据的哪一字节，高位符号扩展； 101表示数据来自MMU，此时为LHU操作，根据alu_result的低2位决定使用MMU数据的哪一字节，高位零扩展； 110表示数据来自CP0寄存器；	
rd_addr	in	std_logic_vector(4 downto 0)
	rd寄存器编号，要求WB阶段上升沿之前准备好。	
rt_addr	in	std_logic_vector(4 downto 0)
	rt寄存器编号，要求WB阶段上升沿之前准备好。	
write_addr	out	std_logic_vector(4 downto 0)
	写入通用寄存器组的地址，请在每一时钟上升沿使用。保持到下一WB阶段时钟上升沿。 对于IP core的片内RAM模块组成的通用寄存器组，此信号请直接接至通用寄存器组。	
write_value	out	std_logic_vector(31 downto 0)
	写入通用寄存器组的数据，请在每一时钟上升沿使用。保持到下一WB阶段时钟上升沿。 对于IP core的片内RAM模块组成的通用寄存器组，此信号请直接接至通用寄存器组。	
write_enable	out	std_logic
	写入通用寄存器组的使能，请在每一时钟上升沿使用。保持到下一WB阶段时钟上升沿。 对于IP core的片内RAM模块组成的通用寄存器组，此信号请直接接至通用寄存器组。	
pc_op	in	std_logic_vector(2 downto 0)
	非ERET指令时，选择新PC的控制信号 00时，选PC+4，即RPC； 01时，根据比较是否成立进行选择，成立则跳转到PC+4+immediate，即RPC+immediate；否则跳转到RPC。 10时，跳转到immediate << 2，对应某些J、JAL语句； 11时，跳转到ALU的计算结果，对应某些JALR、JR语句； 要求WB阶段上升沿之前准备好。	
comp_op	in	std_logic_vector(2 downto 0)
	比较跳转时的比较控制信号，一般对应B系列指令。跳转条件如下： 000时，条件为rs的值等于rt的值； 001时，条件为rs的值>=0； 010时，条件为rs的值>0；	

	011时，条件为rs的值<=0； 100时，条件为rs的值<0； 101时，条件为rs的值不等于rt的值； 其他情况，恒为非； 要求ALU阶段上升沿之前准备好。	
rs_value	in	std_logic_vector(31 downto 0)
	rs的值，要求ALU阶段上升沿之前准备好。	
rt_value	in	std_logic_vector(31 downto 0)
	rt的值，要求ALU阶段上升沿之前准备好。	
imme	in	std_logic_vector(31 downto 0)
	指令中包含的立即数，要求WB阶段上升沿之前准备好。	
PcSrc	out	std_logic_vector(31 downto 0)
	非ERET指令时，新PC值。 WB阶段上升沿之后可读。	

图 5: 写回模块端口示意图



2.5.2 内部实现

WB文件包含两个模块，即WB模块与PC模块。WB模块在WB阶段上升沿运行。根据输入的控制信号，选择是否写入以及写入的寄存器编号的来源。同时，根据输入的控制信号，选择从哪里获得写入数据。非扩展的情况比较简单，直接将写出数据赋为相应输入值即可。对于扩展的情况，需要根据alu_result选择使用mmu_value的哪一部分作为写入数据的低位。 对于符号扩展的情况，扩展方法是根据有效位最高位决定扩展位写全0或写全1。

PC模块在ALU阶段上升沿进行比较。比较方式使用compare_op控制；在WB阶段，根据pc_op控制对输出PC的赋值。其中，如果输出PC需要由比较结果决定，使用ALU阶段的比较结果控制输出PC的值。这一步输出的PC还需在之后使用别的控制信号决定是否用于IF。

WB模块状态跳转：ALU阶段由ALU模块处理。WB阶段，无条件跳转至IF阶段。

WB模块异常触发：无。

2.6 MMU模块

2.6.1 端口说明

端口名	端口方向 端口类型
	端口描述
clk	in std_logic
	CPU时钟信号
rst	in std_logic
	初始化信号，在CPU启动时使用。
state	in status（自定义状态集合）
	说明：CPU状态机信号
if_addr	in std_logic_vector(31 downto 0)
	说明：取指令地址 来源：IFetch模块 到达时间：取指令时钟上升沿之前
instruction	out std_logic_vector(31 downto 0)
	说明：取指令阶段得到的32位指令 产生时间：访存阶段结束之后，根据ready位进行判断
virtual_addr	in std_logic_vector(31 downto 0)
	说明：访存阶段的虚拟地址 来源：MEM模块 到达时间：访存阶段时钟上升沿之前
data_in	in std_logic_vector(31 downto 0)
	说明：访存阶段的写入数据 来源：MEM模块 到达时间：访存阶段时钟上升沿之前
read_enable	in std_logic

	说明：内存读使能，需要进一步处理 来源：MEM模块 到达时间：访存阶段时钟上升沿之前	
write_enable	in	std_logic
	说明：内存写使能，需要进一步处理 来源：MEM模块 到达时间：访存阶段时钟上升沿之前	
data_out	out	std_logic_vector(31 downto 0)
	说明：访存阶段输出的结果 产生时间：访存阶段结束之前，在ready位置1的时候	
ready	out	std_logic
	说明：标志位，访存是否结束 产生时间：访存阶段结束之前	
serial_int	out	std_logic
	说明：串口中断信号，输出到异常模块 产生时间：外部中断，任意时间均可产生	
exc_code	out	std_logic_vector(2 downto 0)
	说明：异常信号，输出到异常模块 产生时间：访存阶段第一个下降沿，下一个下降沿就被清空 000 无异常产生 001 TLB修改异常 010 TLB缺失（读） 011 TLB缺失（写） 100 地址不对齐（读） 101 地址不对齐（写）	
tlb_write_struct	in	std_logic_vector(66 downto 0)
	说明：TLB写结构，包含一个TLB表项所需的所有信息 来源：CP0模块 到达时间：始终从CP0模块连接到MMU，始终处于可读状态	
tlb_write_enable	in	std_logic
	说明：TLB写使能，判断是否写入 来源：指令解码模块 到达时间：指令解码阶段上升沿之后	
align_type	in	std_logic_vector(1 downto 0)

	说明：地址对齐方式，配合地址不对齐异常使用 来源：指令解码模块 到达时间：指令解码阶段上升沿之后	
to_physical_addr	out	std_logic_vector(23 downto 0)
	说明：给物理访存模块的地址，包括了访存类型以及地址 产生时间：访存阶段第一个下降沿之前	
to_physical_data	out	std_logic_vector(31 downto 0)
	说明：给物理访存模块的数据，写内存或串口时需要 产生时间：访存阶段第一个下降沿之前	
to_physical_read_enable	out	std_logic
	说明：给物理访存模块的读使能，异常状态下直接置0 产生时间：访存阶段第一个下降沿之前	
to_physical_write_enable	out	std_logic
	说明：给物理访存模块的写使能，异常状态下直接置0 产生时间：访存阶段第一个下降沿之前	
from_physical_data	in	std_logic_vector(31 downto 0)
	说明：从物理访存模块返回的数据，经过处理直接输出 产生模块：物理访存模块 到达时间：访存时间长度不定，在ready位置1之后	
from_physical_ready	in	std_logic
	说明：从物理访存模块返回的状态位，说明访存是否完成 产生模块：物理访存模块 产生时间：物理访存结束之后	
from_physical_serial	in	std_logic
	说明：串口状态，如果串口有数据则为1 产生模块：物理访存模块 产生时间：串口为外部中断，因此可能在任意时间产生	

2.6.2 内部实现

MMU模块作为真实访问物理内存阶段的预处理阶段，完成对内存的读写控制，完成TLB查询，充填TLB，抛出TLB异常等操作，并且检查地址是否对齐，根据指令类型抛出地址不对齐异常。

MMU模块的初始化决定CPU第一次取指令的地址，需要保证按下rst开关后，to_physical_addr初始化为ROM的起始地址。

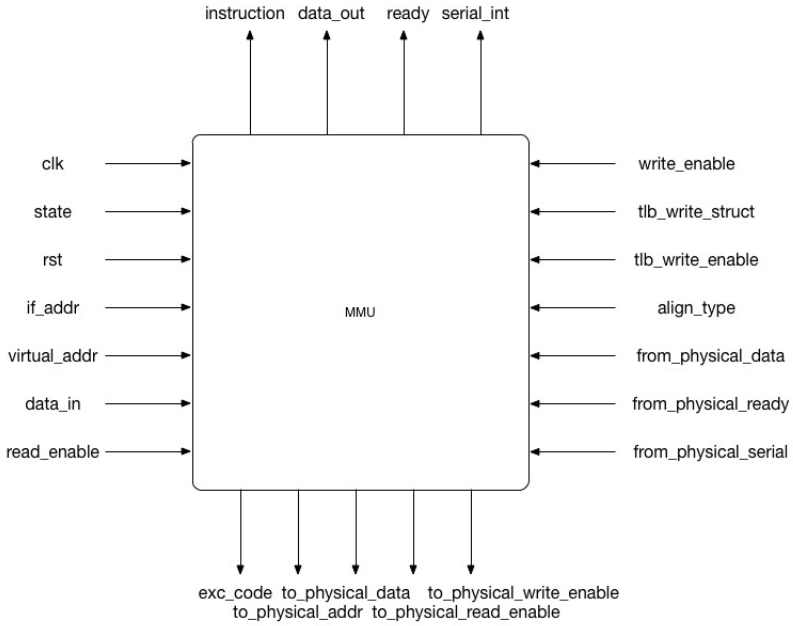
时钟控制：访存阶段上升沿：虚拟地址到物理地址的转换，检查地址是否对齐访存阶段下降沿：抛出TLB、地址不对齐等异常，将访存信号传递给物理内存，开始实际访存访问。

读写控制：

1. 从MEM模块输入的读写使能信号需要经过处理后输出到物理访存模块访存条件为上层有使能信号，处于合适的CPU阶段地址转换不出现异常，物理访存处于停止状态

2. 如果当前state为取指令阶段，则一定为内存读取状态。如果当前为第一访存阶段，因为SB指令读写使能均为1，所以需要进行。如果mem_read_enable为1则进行读操作，否则如果mem_write_enable为1则进行写操作。如果当前为第二访存阶段，则一定为SB指令，进行写操作。

图 6: MMU模块端口示意图



地址映射：根据实际访问的地址值进行判断。只对部分地址（[0x20000000 ~0x80000000] 和 [0xC0000000 ~0xFFFFFFFF]）进行映射，其他地址不经转换，直接访问。

异常与中断：MMU模块可能出现5种异常，一种中断异常信号输出为exc_code, 中断信号输出为serial_int，输出到异常处理模块。异常、中断信号产生时间为访存的时钟下降沿，在下一个时钟下降沿就会被清空，因此需要异常处理模块及时记录异常信息异常中断产生之后，物理内存访问的两个enable为全都会被强制置零，保证在异常状态下不会产生实际的访存操作。

1. 地址不对齐异常两种：根据指令类型判断当前指令是否需要对齐地址进行访问。（LB/SB/LBU指令不需要对齐地址，LHU指令最后一位对齐）根据访存地址后两位判断地址是否对齐。根据读写使能最终确定两种异常中的某一种。

2. TLB异常共三种：均在TLB查找结束之后生成。TLB缺失异常信号，根据查找的结果进行判断，如果为全零则根据度写使能触发异常。TLB修改异常，根据TLB查找结果的D标志位进行判断。

3. 串口中断：直接将物理访存模块的中断信号输出，中断信号产生后将会一直保持，同时由于EXL位屏蔽中断，并不会产生实际的影响直到对串口进行读取操作之后，串口中断才消除。

TLB表查找：向勇老师的课件中有TLB的详细实现方案。为保证效率，此次实验中采用了全相连的TLB设计，但即使不采用并行查找，换为串行查找的策略，在时间上也不会产生影响，且实现方便很多。

1. 采用for_generate/if_generate语句生成TLB查找表。实际效果相当于将输入的虚拟地址高19位复制16份，同时与16个EntryHi进行比较，结果为16位std_logic_vector，其中只有1位为1，其余15位为0。

2. 再利用for_generate/if_generate语句生成TLB结果暂存表，为32*21矩阵32行对应一个16个TLB表项全部Lo部分，21为20位物理地址加一位D标志位。

3. 利用并行比较结果和虚拟地址最低位，共同对暂存表进行与操作，由于其中包含大量的0，最终只选择出1*21的std_logic_vector向量，即为20位物理地址与一位D标记位。

4. 如果查找到TLB查找到物理地址，且D标记位有效，则TLB查找成功，否则查找失败。

TLB表重填：

1. TLB数据来源来自于CP0寄存器，共需要Index、PageMask、EntryHi、EntryLo0、EntryLo1五个寄存器的数值。CP0模块与MMU模块之间始终有以上5个寄存器的连线，始终能够获得CP0寄存器的最新值。

2. TLB充填发生在TLBWI指令的执行阶段，在时钟上升沿、state为执行阶段、tlb_enable使能信号为1的情况下，重填TLB表项。

多次访存问题：

1. 此次实验中访存可能会持续多个时钟周期，因此只能采用访存第一个周期时的信号，之后几个周期中的信号可能会发生改变，应算作无效信号，不予处理。

2. 访存的虚拟地址，需要在IF或者MEM阶段的第一个始终上升沿进行锁存，否则PCmmu信号可能在下一个上升沿发生变化，产生异常情况。

3. 给物理访存层面的两个使能信号，to_physical_read_enable 与 to_physical_write_enable，在访存的第一个下降沿赋值，在第二个下降沿清零，防止出现多次访存的情况。

2.7 CP0模块

2.7.1 端口说明

端口名	端口方向	端口类型
	端口描述	
clk	in	std_logic
	CPU时钟信号	
state	in	status（自定义状态集合）
	CPU当前状态	
normal_cp0_in	in	std_logic_vector(37 downto 0)
	指令引发的CP0读写操作的输入，格式上，37位为写使能，36-32位为地址，31-0位为数据。 读写均在时钟上升沿触发，因此要求数据在时钟上升沿之前准备好。 状态方面，读操作发生在ID阶段的上升沿，写操作发生在ALU阶段的上升沿。	
bad_v_addr_in	in	std_logic_vector(31 downto 0)

	异常发生时写入bad_v_addr_in的数据，要求数据在时钟上升沿之前准备好。使能为interrupt_start_in。	
entry_hi_in	in	std_logic_vector(19 downto 0)
	异常发生时写入entry_hi_in高20位的数据，要求数据在时钟上升沿之前准备好。使能为interrupt_start_in。	
interrupt_start_in	in	std_logic
	异常写入的使能，控制异常数据的写入，并将status(1)置1.	
cause_in	in	std_logic_vector(4 downto 0)
	异常发生时写入cause的6-2位的数据，要求数据在时钟上升沿之前准备好。使能为interrupt_start_in。	
epc_in	in	std_logic_vector(31 downto 0)
	异常发生时写入epc的数据，要求数据在时钟上升沿之前准备好。使能为interrupt_start_in。	
eret_enable	in	std_logic
	eret的使能信号，将status(1)置0.优先于interrupt_start_in起效。 请注意，这一数据应当在ALU阶段上升沿之前准备好。	
compare_init	in	std_logic
	时钟中断恢复的使能。	
addr_value	out	std_logic_vector(31 downto 0)
	normal_cp0_in读操作时读出的数据，ID阶段上升沿之后起效，下一ID阶段上升沿之前均不变。 初始值为全0.	
all_regs	out	std_logic_vector(1023 downto 0)
	即时输出全部CP0寄存器的值，CP0寄存器数值被修改的时间内不保证数值稳定。	
compare_interrupt	out	std_logic
	clock寄存器与compare寄存器数值相同之后被置1；修改compare寄存器的值后置0且该周期不比较clock寄存器与compare寄存器的值。缺少将此值恢复为0的信号。变为1后，若不手动恢复为0或者修改compare寄存器，则1保持。此输出值被检测到为1时触发中断，任一上升沿均可检测，不需太早处理。	

2.7.2 内部实现

CP0寄存器编号严格遵照《See MIPS Run》中的标准定义。

时钟触发。检测到cp0_e为0时，对内部值进行初始化。否则，时钟上升沿时根据state进行相应操作。若state为ID，根据输入地址进行读取操作，需要将normal_cp0_in锁存，该信号只能在一个时钟上升沿中保持不变。若state为ALU，首先本次指令是否为ERET，即检查eret_enable，如果为1则将status(1)置0。然后检测本次指令是否为写CP0，即检查normal_cp0_in(37)，如果需要写则进行CP0写入。注意，此时不进行clock的自增。对于其他state，检查interrunp_start_in是否为1，如果为1说明要进行异常信息的写入。

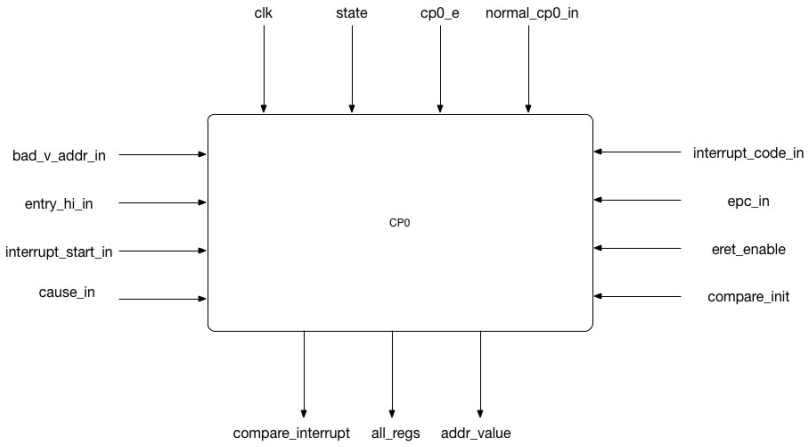
除此之外，还要进行clock寄存器与compare寄存器的比较，compare寄存器的原有值被保存，如果修改，则更新原有值，并将时钟中断置0；否则，比较两寄存器的值，如果相等则置1，否则不变。因此两寄存器第一次相等之后，触发时钟中断。一定要注意，需要将时钟中断恢复的信号！

CP0的Status寄存器有较多内容与软件硬件接口相关，在此单独进行说明。

Status寄存器在此次实验中有4个相关位需要设置：

1. EL：通过软件进行设置，与中断屏蔽相关。硬件上只进行检测，判断是否可以触发中断，该位功能的详细说明请参考《See MIPS Run》。
2. EXL:硬件进行设置，进入异常处理时设置为1，退出异常处理时设置为0。硬件上进行检测，判断是否可以触发中断。
3. KSU：通过软件进行设置，区分用户态和内核态。硬件上可以不作处理，但是如果需要检测用户态访问内核态地址的错误，可以使用这一位进行判断
4. MASK：通过软件进行设置，屏蔽某一种中断。硬件上可以不作处理，但是如果需要单独屏蔽某一种中断，可以使用这一位进行判断。

图 7: CP0模块端口示意图



CP0模块状态跳转：无。

CP0模块异常触发：任何时候检查到compre_interrupt为1均触发异常。此信号会保持且无需即时相应，在适应的时候触发异常即可。

2.8 异常处理模块

2.8.1 端口说明

端口名	端口方向	端口类型
	端口描述	
clk	in	std_logic
	CPU时钟信号	
state	in	status
	自定义状态集合	
exception_e	in	std_logic
	exception模块使能信号	
mmu_exc_code	in	std_logic_vector(2 downto 0)
	来自MMU的异常信号，表示TLB_MODIFIED、TLB_L、TLB_S、ADE_L、ADE_S异常。要求exception阶段时钟上升沿之前保持。	
serial_int	in	std_logic
	来自串口的异常信号。要求exception阶段时钟上升沿之前保持。	
compare_interrupt	in	std_logic
	来自CP0的时钟中断信号。要求exception阶段时钟上升沿之前保持。	
id_exc_code	in	std_logic_vetor(1 downto 0)
	来自ID的异常信号，表示SYSCAL,RI异常。要求exception阶段时钟上升沿之前保持。	
pc_in	in	std_logic_vector(31 downto 0)
	本指令的PC,来自CPU模块。要求exception阶段时钟上升沿之前保持。	
v_addr_in	in	std_logic_vector(31 downto 0)
	目前的访存虚拟地址，来自MMU。要求exception阶段时钟上升沿之前保持。	
old_entry_hi	in	std_logic_vector(19 downto 0)

	旧的entry_hi，用于entry_hi不变的情况，来自CP0。要求exception阶段时钟上升沿之前保持。	
old_interrupt_code	in	std_logic_vector(5 downto 0)
	旧的中断号，用于中断号不变的情况，来自CP0。要求exception阶段时钟上升沿之前保持。	
bad_v_addr_out	out	std_logic_vector(31 downto 0)
	bad_v_addr输出值，交给CP0模块进行写入，可以保持到下次改变。	
entry_hi_out	out	std_logic_vector(19 downto 0)
	entry_hi输出值，交给CP0模块进行写入,可以保持到下次改变。	
interrupt_start_out	out	std_logic
	CP0模块的异常写入使能，控制CP0模块开始写入异常信息，可以保持到下一时钟上升沿之前。	
cause_out	out	std_logic_vector(4 downto 0)
	异常号输出值，交给CP0模块进行写入，可以保持到下次改变。	
interrupt_cause_out	out	std_logic_vector(5 downto 0)
	中断号输出值，交给CP0模块进行写入，可以保持到下次改变。	
epc_out	out	std_logic_vector(31 downto 0)
	EPC输出值，交给CP0模块进行写入,可以保持到下次改变	
pc_sel0	out	std_logic
	IF阶段选择PC的pc_sel的0位，若为1表示应选择异常处理向量作为新的PC。可以保持到下一时钟上升沿之前。	

2.8.2 内部实现

exception负责产生、发送异常信息，根据目前的异常号、中断号，从输入值选择适当的异常信息作为写入CP0的数据。

除写入异常信息外，还需将interrupt_start_out,pc_sel0两个控制信号置1，使CP0准备写入以及异常信息保存后跳转到EBase。其他情况下将其置0。

异常处理设计简述：

1. 需要考虑的异常：

   (a) Interrupt，外部中断。包括时钟中断，串口中断。根据操作系统，设定串口中断标号为2，时钟中断编号为7。

(b) TLB Modify，对内存的只读部分进行写操作。

(c) TLBL，读时发生的TLB miss

(d) TLBS，写时发生的TLB miss

(e) ADEL，对非对齐地址进行读操作

(f) ADES，对非对齐地址进行写操作

(g) SYSCALL，系统调用

(h) RI，执行未定义指令

2. 对于以下异常，不予考虑：

(a) 访问未定义的CP0寄存器。未定义的寄存器视作通用寄存器。

(b) 运算溢出。不予处理。

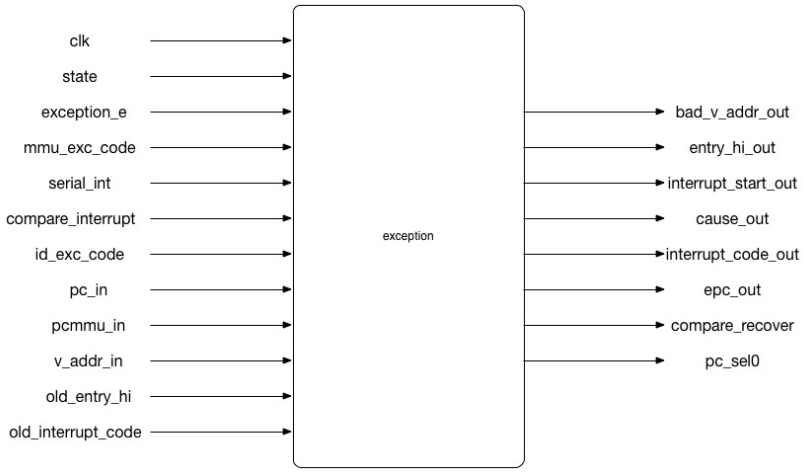
异常编号严格遵照《See MIPS Run》中的定义。

3. 异常处理数据来源（部分异常中未明确提到的信号，请保持原值）：

(a) Interrupt： 由CP0的compare信号与串口的可读信号触发，在IF阶段开始时检查。 异常被处理前信号一直保持，时钟中断被处理后需给CP0模块控制信号消除异常，串口中断被处理后由串口读写部分消除异常。 CP0 status(EXL)即(13)(1)位为'1'时，表示处于异常处理中，不触发外部异常。 检查到异常时，记录异常号、中断号，bad_v_addr取当前指令的地址，EPC取当前指令的地址。

(b) TLB Modify： 由MMU的对应信号触发，发生在MEM阶段，在ID、WB阶段开始时检查。 异常在下一时钟下降沿消除，因此需在要求的上升沿进行检查。 检查到异常时，记录异常号，bad_v_addr取MMU提供的虚拟地址，EPC取当前指令的地址。

图 8: 异常处理模块端口示意图



(c) TLBL：由MMU的对应信号触发，发生在IF阶段或MEM阶段，在ID、WB阶段开始时检查。异常在下一时钟下降沿消除，因此需在要求的上升沿进行检查。检查到异常时，记录异常号，bad_v_addr取MMU提供的虚拟地址，EPC取当前指令的地址，EntryHi高20位取MMU提供虚拟地址。

(d) TLBS：由MMU的对应信号触发，发生在MEM阶段，在ID、WB阶段开始时检查。异常在下一时钟下降沿消除，因此需在要求的上升沿进行检查。检查到异常时，记录异常号，bad_v_addr取MMU提供的虚拟地址，EPC取当前指令的地址，EntryHi高20位取MMU提供虚拟地址。

(e) ADEL：由MMU的对应信号触发，发生在IF阶段或MEM阶段，在ID、WB阶段开始时检查。异常在下一时钟下降沿消除，因此需在要求的上升沿进行检查。检查到异常时，记录异常号，bad_v_addr取MMU提供的物理地址，EPC取当前指令的地址。

(f) ADES：由MMU的对应信号触发，发生在MEM阶段，在ID、WB阶段开始时检查。异常在下一时钟下降沿消除，因此需在要求的上升沿进行检查。检查到异常时，记录异常号，bad_v_addr取MMU提供的物理地址，EPC取当前指令的地址。

(g) SYSCALL：由ID模块的对应信号触发，发生在ID阶段，在ALU阶段开始时检查。异常在下一时钟上升沿消除，因此需在要求的上升沿进行检查。检查到异常时，记录异常号，bad_v_addr取当前指令地址。EPC取当前指令的地址，EPC+4的操作由操作系统完成。

(h) RI：由ID模块的对应信号触发，发生在ID阶段，在ALU阶段开始时检查。异常在下一时钟上升沿消除，因此需在要求的上升沿进行检查。检查到异常时，记录异常号，bad_v_addr取当前指令地址，EPC取当前指令的地址。

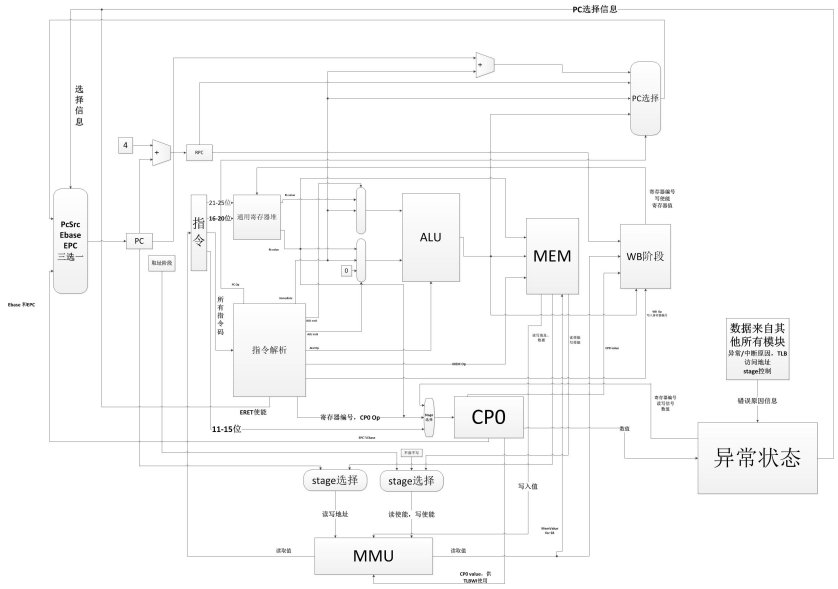
exception模块状态跳转：请无条件跳转至IF阶段。
exception模块异常触发：无。

3 整体设计

3.1 CPU整体设计

CPU元件例化独立的模块，同时对各个模块的异常与状态信息进行收集，决定主状态机的跳转。

图 9: CPU整体设计图



3.2 元件例化

CPU顶层主要的功能为实现例化各个单独模块并进行模块间的互联。按照CPU整体设计图，元件例化每个在模块设计部分所出现的模块，各个模块间的连接线由模块设计部分给出。

3.3 其他实现

除元件例化外，CPU顶层还需实现的功能为主状态机的跳转、时钟分频以及异常信息的处理。

3.3.1 状态机跳转

CPU顶层需要实现主状态机的跳转。
与状态机跳转相关的信号有五个，以下分别对其功能进行说明

信号名	信号类型　信号简介
	信号描述
has_mem1	std_logic　是否有第一访存周期
	由于对PC的修改在WriteBack阶段进行，因此每条指令都必须有写回阶段。为简化实现，也迫使每条指令都有执行阶段，不需要使用ALU的指令在此阶段不做任何操作。 在指令解码的时钟上升沿对指令进行分类，对需要进行访存的指令，has_mem1置1，表示有访存阶段。其他不涉及到访存指令的has_mem置0
has_mem2	std_logic　是否有第二访存周期
	专门为SB指令设计，判断是否有第二访存周期。仅当指令为SB时has_mem2置1，其他情况置0
old_state	status　访存保持状态
	CPU对Ram、Flash、串口的访问进行了统一的封装，因此一次访存的时间可能超过一个时钟周期。old_state用来在访存时间超过一个时钟周期时，保持访存的状态不变。
next_state	status　下一状态
	表示当前状态的下一状态。由于执行和写回阶段是每条指令必须经过的阶段，因此只通过has_mem1和has_mem2两个信号，对是否有访存阶段进行选择。 该信号通过时序逻辑进行控制，每个时钟上升沿根据state进行变化。
state	status　当前状态
	CPU状态机的当前状态，可能的取值为old_state或者next_state或者为异常状态。该信号通过组合逻辑进行控制。 在访存过程中，访存的busy信号持续为1，此时state保持为old_state，保证一次访存结束之后，CPU主状态机再继续跳转。 如果有异常信号被置1，则state变为异常状态。之后由异常处理模块保存异常信息，跳转到异常处理向量，再次进行取指令操作。 其他状态下，state被赋值为next_state，表示正常情况下的状态跳转。

3.3.2　异常处理

CPU顶层收集来自各个模块的异常信息，用来判断状态跳转，其他异常相关工作交给异常处理模块负责。

与异常处理相关的信号有三个，一下分别对其功能进行说明。

信号名	信号类型　信号简介

	信号描述
clock_inter_to_excep	std_logic　时钟中断信号
	初始的时钟中断信号由CP0模块产生，当CP0中status寄存器的EXL位为0且EL为为1时，中断有效。 同时，由于外部中断并不需要在产生的时候立刻被处理，因此我们选择在写回阶段，对中断信号进行判断，保证这一条指令正常执行完毕后，再对时钟中断进行处理。
serial_inter_to_excep	std_logic　串口中断信号
	初始的串口中断信号由访存模块产生。其他处理方式与时钟中断相同。
excep	std_logic　异常中断信号
	表明是否发生异常或中断，用来决定CPU主状态机的跳转。 正常情况下所有模块发送来的异常信号均为0，因此将所有异常位做逻辑或，即为excep信号。

3.3.3　时钟分频

访存模块始终工作在$25MHz$的高频下， 分频后得到CPU时钟的工作频率。

CPU单独运行时采用四分频，工作在$6.25MHz$。 调试模式与CPU单独运行时环境完全相同， 也为$6.25MHz$。

4 软件硬件接口

在实际的开发过程与ucore的调试过程中，我们逐渐发现了一些在设计过程中没有注意到的软件硬件接口问题。为记录我们开发的过程，同时也为了后续完成此实验的同学能够有所参考，我们特别加入软件硬件接口这部分内容。

4.1 异常中断相关

4.1.1 寄存器与中断异常编号

CP0寄存器编号和异常编号与标准MIPS相同，具体可以参考《See MIPS Run》，原实验指导文档中的寄存器编号有误。

异常编号与标准MIPS相同，具体可以参考《See MIPS Run》，指导文档该部分也是正确的，可以参考。

中断在此次实验中实现了时钟中断与串口中断，其编号没有统一标准，各种实现版本均不同。此次实验以刘亚宁学长的os_lab为主要参考，因此串口中断标号为2，时钟中断编号为7。

4.1.2 时钟中断

标准MIPS中可以通过status寄存器中的EXL与EL位进行整体的中断屏蔽，还可以通过MASK进行单独的某个中断的屏蔽。在刘亚宁学长的os_lab中，启动过程中是通过MASK位进行屏蔽，保证不会被时钟中断所打断。

此次实验中在硬件上没有提供对MASK位的支持，因此选择在操作系统上进行修改。CP0的Compare寄存器初始化为0xFFFFFFFF，保证初始化结束之前一定不会触发时钟中断。初始化完成后增加一次对clock_intr函数的调用，将Compare寄存器重填为设定的数字，之后即可触发时钟中断。

标准MIPS时钟中断还应该在硬件上实现“读Count写入Compare”功能，此问题已经在os_lab中通过软件方式解决，不需要另作处理。

4.1.3 串口

从lab3开始，由于刘亚宁学长的代码支持了在实验板与qemu两个平台上的运行，两平台的串口地址不同。因此，在编译操作系统前需要先调用to_thin脚本，将操作系统转换为实验板上的版本。

使用贾开学长推荐的开源串口代码，实现了稳定的传输。

串口实现为一个数据位加一个状态位。状态位第0位为是否可写，第1位为是否可读。读写串口前都会先检查是否可读可写，之后才进行真正的读写操作。此次实验中，串口的写标记为始终可写，在串口写的过程中阻塞CPU，写过程结束之后CPU才会继续执行后续的指令。串口的读标记直接使用开源串口工具的读标记，并输出至CPU顶层模块，用来触发串口中断。

串口的读写使用的是LW和SW指令，对32bit进行操作，但实际上之后低8位是有效的。

4.1.4 系统调用

系统调用syscall指令会触发一个异常，异常处理结束后应该跳转到下一条指令继续执行。其中EPC加4的操作是由操作系统实现的，不需要提供硬件支持。因此，对于所有中断和异常的处理，其EPC均为当前指令的地址。

4.1.5 异常处理向量

刘亚宁学长的异常处理向量只有唯一的一个入口0x80000180，在初始化阶段直接写入EBASE寄存器之后就不需要再进行修改了。异常处理的初始化完全由操作系统完成，在0x80000180存入一条jump指令，跳转值alltraps函数，硬件上不需要任何的特殊处理。

4.1.6 异常定义

操作系统对某些异常并没有进行处理，而是直接停止执行后进入debug_monitor模式，这些异常包括AdEL、AdES、TLB_Modify。

在lab5的check_swap部分中曾出现过TLB_Modify的异常，操作系统生成的两个物理页的的D标记有问题，原因不明。最后在硬件上直接取消了异常，操作系统即可正常运行通过check_swap，并且之后的代码也能够正常运行，因此直到最后的lab8仍然采用的是这种方式。

AdEL与AdES两种异常，指导文档中给出的定义是地址不对齐，但实际上内存的跨界访问（用户态访问内核态）也是通过这两个异常表示的。

4.1.7 Status寄存器

Status寄存器在此次实验中有4个相关位需要设置：

1. EL：通过软件进行设置，与中断屏蔽相关。硬件上只进行检测，判断是否可以触发中断，该位功能的详细说明请参考《See MIPS Run》。
2. EXL:硬件进行设置，进入异常处理时设置为1，退出异常处理时设置为0。硬件上进行检测，判断是否可以触发中断。
3. KSU：通过软件进行设置，区分用户态和内核态。硬件上可以不作处理，但是如果需要检测用户态访问内核态地址的错误，可以使用这一位进行判断
4. MASK：通过软件进行设置，屏蔽某一种中断。硬件上可以不作处理，但是如果需要单独屏蔽某一种中断，可以使用这一位进行判断。

4.2 其他

4.2.1 操作系统组成

此次实验中操作系统bootloader部分使用贾开学长的代码（与Flash访问方式有关，见第二点说明），其余均在刘亚宁学长os_lab基础上进行修改，具体修改内容之后有详细叙述。

4.2.2 Flash访问

因为Flash数据线只有16条，所以实际上Flash连续的4个byte中只有2个为有效数据，因此产生了两种Flash访问方式如下：

一种方式为每次访问Flash的4个byte，得到32位数据中只有低16位有效。之后在软件层进行控制，连续访问两次，将结果移位拼接得到32位有效数据。另一种方式为每次访问Flash在硬件上访问8个byte，硬件上将两次得到的数据进行拼接，对操作系统层提供与RAM访问相同的接口。

此次实验中我们使用第一种方式。因此在bootloader中使用贾开学长的代码，其中包含了将访存结果移位拼接的实现。除bootloader之外的操作系统使用刘亚宁学长的代码，这些代码需要第二种Flash访问方式进行支持。

lab1到lab7不需要进行任何修改，因为所有Flash访问操作均是在bootloader部分完成。在lab8中由于涉及到文件系统sfs.img的加载，需要修改lab8/kern/fs/devs/dev_disk0.c中的disk0_read_blks_nolock函数，将Flash起始地址与DISK0_BLKSIZE均乘2，之后通特别外实现的memcpy_flash将sfs.img加载到内存中。

4.2.3 初始化

bootloader运行前需要对硬件做一次初始化，主要是各种寄存器的初始值设置。在此次实验中利用实验板的rst开关进行。

CP0寄存器Compare寄存器全部设置为1，保证操作系统初始化完成前不会触发时钟中断。EBASE寄存器设置为0x80000180，为异常处理向量起始地址。其余CP0寄存器均设置为0。

访存地址设置为0x90000000，为bootloader的起始地址。通用寄存器全部设置为0。各个模块的辅助信号按照需求进行初始化。

4.2.4 ROM

操作系统需要一块ROM保存，bootloader中的指令，需要保证断电不失。Xilinx提供的IPcore中可以实现片内的ROM，但是使用方法比较复杂。

另一种实现方式，也是本次实验中使用的方式，是直接用VHDL语言生成一个std_logic_vector的常量数组，保存每一条指令。之后直接通过下标进行访问即可。

5 问题与解决

在CPU的测试阶段中，我们发现了很多在编码阶段比较难预想到的问题。我们将这些问题记录在这里，记录我们的调试过程，同时也希望能对未来完成此实验的同学有所帮助。

5.1 编译工具

此次实验中采用的编译工具为mips-linux-gnu工具链，版本为(Sourcery CodeBench Lite 2014.05-27)2.24.51.20140217。编译器版本不同可能导致代码段组装顺序不同，进而可能产生问题。

问题现象：在实验中运行lab1时，在bootloader运行结束后，运行kern_boot函数，设置栈指针sp的初始值为kern_boot。但是kern_boot函数位于代码段的中央由此导致数据段将代码段覆盖。原因为刘亚宁学长使用的编译器将kern_boot布局在代码段的顶端，而我们的编译器将kern_boot布局于中间部分。

解决方案：修改了kern_boot函数，设置栈指针为代码段顶端的kern_init函数。

5.2 PC选择信号

实验中PC共有三种可能来源，正常运行下PC的计算结果，触发异常后的EBASE，异常返回时的EPC。控制将EPC赋值给PC的信号eret_enable由指令解码模块产生。原实现方案中，该信号在eret指令解码阶段产生，保持到下一条指令的解码阶段。但是在lab5运行时出现错误，不能正常运行。

后发现，lab5中syscall触发异常，在处理过程中将EPC改为用户态程序的入口，eret指令后PC即为用户态程序的入口。但是下一条指令取指阶段会发生TLBmiss，立即进入异常处理，并没有进入解码阶段，导致eret_enable信号并没有被消掉，后续指令无法正常执行。

解决方案：修改解码模块，将eret_enable信号在指令的取指阶段消除。

5.3 MMU访存地址

原实现方案中，在访存和取指阶段的上升沿，MMU模块对访存的地址进行锁存。后续的TLB查找，物理地址的生成均以此地址为基础。

但在实际访存过程中，由于整合了多种访存方式，需要进行更多的判断，因此访存会持续多个时钟上升沿。在取指阶段由于并没有进行PCmmu的锁存，因此可能发生变化，相当于进行了两次地址的锁存。虽然可以控制真实的访存过程只发生第一次，但是对异常地址的判断可能进行多次，导致指令运行出现错误。

解决方案：在访存和取指阶段的上升沿，而且还没开始进行真正的访存时（一定是第一个上升沿）才对地址进行锁存。

5.4 TLB查找策略

TLB表项含有两个EntryLo，根据虚拟地址的第20位选择命中其中一个地址。原实现方案参照贾开学长的代码，但实际运行中发现EntryLo命中错误，本应命中与Lo0的页命中在了Lo1。

原因可能为TLBWI时的写入策略不同，两个EntryLo寄存器写入TLB的位置不同。

解决方案：在向勇老师的课件中发现了TLB的详细实现方案，按照该方案进行了调整，TLB运行正常。之后的实现可以参照该课件进行实现。

此外，原实现方案为保证效率，TLB采用了全相连映射的并行查找策略，利用VHDL的generate语句生成。但是由于VHDL不支持Verilog中的一些语法特性，导致代码的实现比较繁复冗余。后发现使用串行判断并不会过多地影响CPU速度，因此之后的实现可以直接采用串行判断的方法，代码比较简洁。

5.5 寄存器写入

CP0寄存器需要在ID阶段的上升沿根据访存得到的指令，确定读取寄存器的位置，和写入寄存器的位置。

原实现方案中，在CP0模块中没有对两个位置进行锁存。由于访存模块返回的数据可能在一条指令的不同阶段发生变化，由此导致CP0寄存器通过mtc0写入的过程中出现错误。

解决方案:在CP0模块中增加锁存器，在ID阶段的上升沿就锁存读写两个位置，保证写入过程的正确。

在通用寄存器中仍然存在类似问题，解决方案为将ID模块生成的寄存器写回的使能信号发送给WB模块，通过WB模块控制写入信号只保持一个阶段。

5.6 访存计数

需要将多种访存整合成为统一的接口，需要利用一些时钟上升沿进行访存类型等信息的判断。因此需要更长的访存时间，有可能超出一个时钟上升沿。

原实现方案为在访存模块加一个busy为表示正在进行访存，此时CPU主状态机保持原来的值。但是在某些情况下可能出现连续多次访存的情况。

解决方案：引入一个counter信号，当状态机发生变化时清零，表示新的访存阶段开始。状态机不变时保持为1，表示还是原来的访存。只有当counter为1的时候读写使能信号均置为0，表示可以进行访存。由此保证真实的访存操作只进行一次。

5.7 其他

实现过程中参考了刘亚宁学长暑期的实验进展报告，比较方便地避免了其中提到的很多问题。

其中比较重要的是J/JAL指令的实现，实验指导文档上的说明有误。以及在ucore未实现命令的避免部分的TICK_NUM问题。

6 调试工具

在此次实验的扩展部分，我们完成了基于串口的调试工具。

通过PC端发送命令到实验板，可以完成设置断点、启动、继续运行、单步调试等多种功能。并且可以在PC端查看寄存器的值，查看各个模块间传递的信号。

调试工具的使用方法与命令行下的GDB工具十分相似，有GDB使用经验的人可以轻松上手。

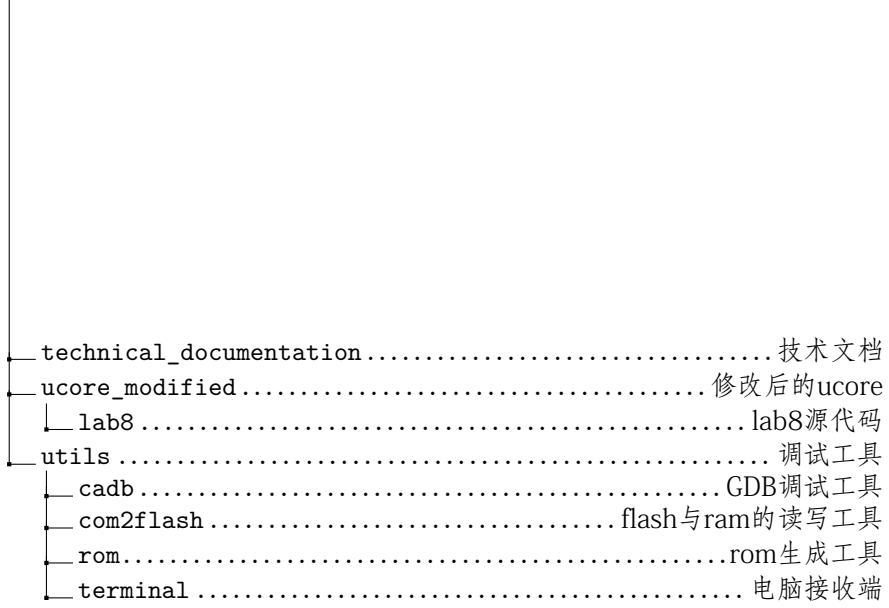
此外，调试工具在设计时就考虑到了复用的问题，因此与原有VHDL代码之间的耦合度非常小，只需要在原有代码上稍作修改即可运行。修改部分也有代码自动生成工具予以辅助，在PC端配置config文件后即可自动生成需要添加的代码。

调试工具的详细设计思路请参见调试工具设计文档。

调试工具的具体使用方式请参见调试工具使用手册。

7 文件结构

```
/
├── bin ..... 二进制文件
│   ├── cacpu.bit ..... CPU，烧入FPGA
│   ├── cpld.jed ..... 烧入CPLD
│   ├── kernel ..... 操作系统内核
│   ├── sfs.img ..... 文件系统
│   └── cadb.bit ..... 带有Cache Debugger调试工具的CPU
├── design ..... 设计资料
│   └── datapath.xls ..... 数据通路指令信号说明
├── product_requirement_document ..... 需求文档
└── src ..... 源代码
    ├── cacpu ..... Cache CPU
    │   ├── cacpu.xise ..... 工程文件
    │   ├── CP0.vhd ..... CP0模块
    │   ├── Exception.vhd ..... 异常处理模块
    │   ├── IDecode.vhd ..... 指令解析模块
    │   ├── IDecode_const.vhd ..... 指令解析常量
    │   ├── IFetch.vhd ..... 取指令模块
    │   ├── WB.vhd ..... 写回模块
    │   ├── alu.vhd ..... ALU模块
    │   ├── async.v ..... 开源串口异步收发模块
    │   ├── cacpu.ucf ..... 管脚绑定
    │   ├── cacpu.vhd ..... 顶层控制模块
    │   ├── common.vhd ..... 通用常量
    │   ├── flash.vhd ..... Flash操作模块
    │   ├── general_register.vhd ..... 通用寄存器模块
    │   ├── mem.vhd ..... 访存模块
    │   ├── mmu_module.vhd ..... MMU模块
    │   ├── phy_mem.vhd ..... 物理访存模块
    │   ├── ram.vhd ..... RAM读写模块
    │   └── rom.vhd ..... ROM读写模块
    ├── cpld ..... CPLD
    │   ├── cpld.ucf ..... 管脚绑定
    │   ├── cpld.vhd ..... 转发程序
    │   └── cpld.xise ..... 工程文件
    └── cadb ..... Cache Debugger
        ├── com_debug.vhd ..... 调试命令收发解析
        ├── bp_debug.vhd ..... 调试控制模块
        ├── testcpu.vhd ..... 顶层模块
        └── testcpu.ucf ..... 管脚绑定
```


technical_documentation 技术文档
ucore_modified 修改后的ucore
lab8 lab8源代码
utils 调试工具
cadb GDB调试工具
com2flash flash与ram的读写工具
rom rom生成工具
terminal 电脑接收端