软工计原联合项目

设计文档

NonExist 组
张钰晖，杨一滨，周正平

目录

Chapter 1

文档说明

本文档是 NonExist 组软工计原联合项目的设计文档，作为设计文档，将会尽可能详细的覆盖到所有的设计方面和设计细节。

但是，设计文档呈现的是最终版 CPU，因此每个模块此时都已经经历了无数次蜕变，在这个过程中，每个模块的功能越来越多，也越来越复杂。本文档呈现的是最终版的设计，缺少了循序渐进的过程，因此读者初读起来可能感到困难，不建议将此文档作为前期主要参考文档。

本文档正确的使用方式是，开发初期通读全文，站在更高的层面上俯视了解整个项目的设计；开发后期精读细节，实现和完善具体的功能。

设计文档将项目分成了以下部分：

1. **指令流水**：本阶段实现 CPU 五级流水线及绝大部分基本指令。
2. **控制模块**：本阶段实现协处理器和异常处理
3. **内存管理**：本阶段实现内存管理。
4. **外设连接**：本阶段完成外设连接。

设计文档每个章节遵从以下介绍流程：

1. **简介概述**：简单介绍所实现元件的功能。
2. **接口定义**：实现的接口及其含义。
3. **设计细节**：详细而完善的设计思路与细节。

希望本文档能给读者带来裨益。

Chapter 2

指令流水

本章介绍了 MIPS 标准五级流水线的实现，同时实现了绝大部分需要基本指令。

目前的 CPU 设计主要有三类 CPU：单周期 CPU、多周期 CPU 和指令流水 CPU。我们设计的 CPU 是指令流水 CPU，根据 MIPS 规范，将 CPU 分为了五级流水：取指 (if)、译码 (id)、执行 (ex)、访存 (mem)、回写 (wb)。所谓的流水线 CPU 即将指令处理的过程划分为各个阶段，每条指令顺着流水线的阶段顺序处理，最终处理完整条指令。而由于流水线处理的时候互不相干，(比如，当一条指令在译码的时候，另外一条指令可以在取指)，我们的流水线 CPU 在一个时钟里就可以并行的处理五条在不同阶段的指令，提高 CPU 工作的效率。

流水线 CPU 设计主要分为运算部分和控制部分，本章主要侧重运算部分。而运算部分中除了我们上面所说的五级流水以外主要还包括流水线寄存器部分、通用寄存器部分、特定寄存器部分和数据旁路部分。其中，流水线寄存器主要用于衔接各个流水阶段，起到承上启下的数据阶段传递效果；通用计算器和特定寄存器部分是 CPU 中运算部分所需要的各类寄存器；数据旁路部分主要通过数据前推来解决数据冲突。

2.1 pc_reg

2.1.1 简介概述

pc_reg 阶段实现程序计数，是五级流水线的第一级，记录了当前指令地址，同时对下一条指令地址进行计算与选择，是一个简单的时序逻辑电路。

2.1.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	时钟信号
input	wire	1	rst	CPU 外部	复位信号
input	wire	6	stall	ctrl	流水线暂停使能
input	wire	1	tlb_hit	tlb_reg	TLB 是否命中

input	wire	32	physical_pc	tlb_reg	物理地址
input	wire	1	flush	ctrl	流水线清空使能
input	wire	32	new_pc	ctrl	下一指令地址
input	wire	1	branch_flag_i	id	跳转使能信号
input	wire	32	branch_target_addr_i	id	分支跳转地址
output	reg	32	virtual_pc	tlb_reg、if_id	虚拟地址
output	reg	32	pc	CPU 外部	指令地址
output	reg	1	ce	CPU 外部	访存使能信号
output	reg	32	excepttype_o	if_id	异常类型

2.1.3 设计细节

初始时 virtual_pc 指向 0xbfc00000（ROM 起始地址），之后 virtual_pc 每个时钟周期加 4，若上一条指令为分支跳转指令，则 virtual_pc 置为分支跳转地址。同时，生成 excepttype 异常信息归总，本阶段主要用于发现 TLB 缺失异常。

2.2 if_id

2.2.1 简介概述

if_id 用于衔接五级流水线第一阶段 if 和第二阶段 id，在时钟上升沿储存 if 阶段数据，传递给 id 阶段，将 if 阶段所得到的指令传递至 id 阶段进行译码，是一个简单的时序逻辑电路（流水线寄存器）。

2.2.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	时钟信号
input	wire	1	rst	CPU 外部	复位信号
input	wire	32	if_pc	pc_reg	指令地址
input	wire	32	if_inst	pc_reg	指令内容
input	wire	32	if_excepttype	pc_reg	异常类型
input	wire	6	stall	ctrl	流水线暂停使能
input	wire	1	flush	ctrl	流水线清空使能
output	reg	32	id_pc	id	指令地址
output	reg	32	id_inst	id	指令内容
output	reg	32	id_excepttype	id	异常类型

2.2.3 设计细节

当 if 阶段没有被暂停，置 id_pc 为 if_pc，id_excepttype 为 if_excepttype，此时如果没有发生 TLB 缺失异常，置 id_inst 为 if_inst，其余情况下 id_pc、id_inst 和 id_excepttype 均置 0。

2.3 id

2.3.1 简介概述

id 阶段实现指令译码，是五级流水线的第二级，主要是识别指令类型和各字段、读取通用寄存器值、产生流水线控制信号，除此之外，id 阶段还需要实现数据旁路、分支判断处理等，是一个复杂的组合逻辑电路。

2.3.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	rst	CPU 外部	复位信号
input	wire	32	pc_i	if_id	指令地址
input	wire	32	inst_i	if_id	指令内容
input	wire	32	reg1_data_i	regfile	通用寄存器读端口 1 数据
input	wire	32	reg2_data_i	regfile	通用寄存器读端口 2 数据
input	wire	1	ex_wreg_i	ex	旁路信号，EX 阶段是否写回通用寄存器
input	wire	32	ex_wdata_i	ex	旁路信号，EX 阶段写回通用寄存器数据
input	wire	5	ex_wd_i	ex	旁路信号，EX 阶段写回通用寄存器地址
input	wire	8	ex_aluop_i	ex	旁路信号，EX 阶段指令类型
input	wire	1	mem_wreg_i	mem	旁路信号，MEM 阶段是否写回通用寄存器
input	wire	32	mem_wdata_i	mem	旁路信号，MEM 阶段写回通用寄存器数据
input	wire	5	mem_wd_i	mem	旁路信号，MEM 阶段写回通用寄存器地址
input	wire	1	is_in_delay_slot_i	id_ex	当前指令是否位于分支延迟槽中
input	wire	32	excepttype_i	if_id	异常类型
output	reg	1	next_inst_in_delay_slot_o	id_ex	下一条指令是否位于分支延迟槽中

output	reg	1	branch_flag_o	pc_reg	跳转使能信号
output	reg	32	branch_target_addr_o	pc_reg	分支跳转地址
output	reg	32	link_addr_o	id_ex	link 地址
output	reg	1	is_in_delay_slot_o	id_ex	当前指令是否位于分支延迟槽中
output	reg	1	reg1_read_o	regfile	通用寄存器读端口 1 使能
output	reg	1	reg2_read_o	regfile	通用寄存器读端口 2 使能
output	reg	5	reg1_addr_o	regfile	通用寄存器读端口 1 地址
output	reg	5	reg2_addr_o	regfile	通用寄存器读端口 2 地址
output	reg	8	aluop_o	id_ex	ALU 运算类型
output	reg	3	alusel_o	id_ex	ALU 选择类型
output	reg	32	reg1_o	id_ex	ALU 第一个操作数
output	reg	32	reg2_o	id_ex	ALU 第二个操作数
output	reg	5	wd_o	id_ex	通用寄存器写端口地址
output	reg	1	wreg_o	id_ex	通用寄存器写端口使能
output	wire	32	inst_o	id_ex	指令内容
output	wire	1	stallreq	ctrl	id 阶段的暂停请求
output	wire	32	excepttype_o	id_ex	异常类型
output	wire	32	current_inst_address_o	id_ex	当前指令地址

2.3.3 设计细节

ID 阶段将识别指令的操作码和各个字段，根据指令操作码向 EX 阶段传入其所需信号。具体来说, ID 是五级流水的译码阶段, 这一阶段 CPU 会根据具体的指令给出各个使能信号和操作标识。同时, CPU 会在这个阶段访问寄存器取得寄存器形式的操作数推进到下一阶段进行计算，值得注意的是，本部分还接入了从 EX 阶段和 MEM 阶段回接的数据旁路, 用于解决数据冲突, 即对于 EX 阶段和 MEM 阶段的已经完成的计算中，传递计算出来的结果和所需存储的寄存器编号，若与目前所需获取的寄存器编号相同，则从此旁路获取数据。使能信号和操作标识中，大部分的都是有关操作控制的内容，基本上为后续流水准备，顺着流水线推进；少部分内容用于控制器设计和异常处理，其中的 stallreq 变量就是用于 ID 阶段与控制器交互的，当 ID 阶段需要暂停流水线的时候，会通过此信号告知控制器，而 excepttype_o 是在流水线间传递异常信息归总，ID 阶段主要用于发现异常处理返回、指令不合法、系统调用三个异常。

2.4 id_ex

2.4.1 简介概述

id_ex 用于衔接五级流水线第二阶段 id 和第三阶段 ex，在时钟上升沿储存 id 阶段数据，传递给 ex 阶段，将 id 阶段译码阶段传递至 ex 阶段进行算数逻辑运算，是一个简单的时序逻辑电路。

2.4.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	时钟信号
input	wire	1	rst	CPU 外部	复位信号
input	wire	8	id_aluop	id	ALU 运算类型
input	wire	3	id_alusel	id	ALU 选择类型
input	wire	32	id_reg1	id	ALU 第一个操作数
input	wire	32	id_reg2	id	ALU 第二个操作数
input	wire	5	id_wd	id	通用寄存器写端口地址
input	wire	1	id_wreg	id	通用寄存器写端口使能
input	wire	32	id_link_addr	id	link 地址
input	wire	1	id_is_in_delay_slot	id	当前指令是否位于分支延迟槽中
input	wire	1	next_inst_in_delay_slot_i	id	下一条指令是否位于分支延迟槽中
input	wire	32	id_inst	id	指令内容
input	wire	32	id_current_inst_address	id	当前指令虚地址
input	wire	32	id_excepttype	id	异常类型
input	wire	6	stall	ctrl	流水线暂停使能
input	wire	1	flush	ctrl	流水线清空使能
output	reg	8	ex_aluop	ex	ALU 运算类型
output	reg	3	ex_alusel	ex	ALU 选择类型
output	reg	32	ex_reg1	ex	ALU 第一个操作数
output	reg	32	ex_reg2	ex	ALU 第二个操作数
output	reg	5	ex_wd	ex	通用寄存器写端口地址
output	reg	1	ex_wreg	ex	通用寄存器写端口使能
output	reg	32	ex_link_addr	ex	link 地址
output	reg	1	ex_is_in_delay_slot	ex	当前指令是否位于分支延迟槽中
output	reg	1	is_in_delay_slot_o	ex	下一条指令是否位于分支延迟槽中
output	reg	32	ex_inst	ex	指令内容
output	reg	32	ex_current_inst_address	ex	指令地址
output	reg	32	ex_excepttype	ex	异常类型

2.4.3 设计细节

与之前的 IF/ID 类似，ID/EX 的主要目的也是流水线寄存器，用于缓存来自 ID 阶段的操作信息和使能信号，在下一个时钟周期再推进到 EX 阶段。同时，本阶段也同时对暂停和流水清空作出处理。

通过查看控制器发来的暂停向量，根据 ID 阶段是否暂停来控制是否插入空泡；通过查看控制器发来的清空使能信号，判断是否要清空流水线的内容。当然，大多数情况下，会将 ID 阶段的指令推进到 EX 阶段。

2.5 ex

2.5.1 简介概述

ex 阶段实现算数逻辑运算，是五级流水线的第三级，主要是进行各种算数逻辑运算，例如加法、减法、乘法、移位、与或非等操作，除此之外，ex 阶段还需要实现数据旁路、分支判断处理等，是一个复杂的组合逻辑电路。

2.5.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	rst	CPU 外部	复位信号
input	wire	8	aluop_i	id_ex	ALU 运算类型
input	wire	3	alusel_i	id_ex	ALU 选择类型
input	wire	32	reg1_i	id_ex	ALU 第一个操作数
input	wire	32	reg2_i	id_ex	ALU 第二个操作数
input	wire	5	wd_i	id_ex	通用寄存器写端口地址
input	wire	1	wreg_i	id_ex	通用寄存器写端口使能
input	wire	32	inst_i	id_ex	指令内容
input	wire	32	hi_i	hilo_reg	HI 寄存器读出数据
input	wire	32	lo_i	hilo_reg	LO 寄存器读出数据
input	wire	32	wb_hi_i	mem_wb	旁路信号,WB 阶段写回 HI 寄存器数据
input	wire	32	wb_lo_i	mem_wb	旁路信号，WB 阶段写回 LO 寄存器数据
input	wire	1	wb_whilo_i	mem_wb	旁路信号, WB 阶段是否写回 HILO 寄存器
input	wire	32	mem_hi_i	ex_mem	旁路信号，MEM 阶段写回 HI 寄存器数据
input	wire	32	mem_lo_i	ex_mem	旁路信号，MEM 阶段写回 LO 寄存器数据
input	wire	1	mem_whilo_i	ex_mem	旁路信号，MEM 阶段是否写回 HILO 寄存器
input	wire	32	link_addr_i	id_ex	LINK 地址
input	wire	1	is_in_delay_slot_i	id_ex	当前指令是否位于分支延迟槽中

input	wire	1	mem_cp0_reg_we	ex_mem	旁路信号, MEM 阶段是否写回 CP0 寄存器
input	wire	5	mem_cp0_reg_write_addr	ex_mem	旁路信号, MEM 阶段写回 CP0 寄存器地址
input	wire	32	mem_cp0_reg_data	ex_mem	旁路信号, MEM 阶段写回 CP0 寄存器数据
input	wire	1	wb_cp0_reg_we	mem_wb	旁路信号, WB 阶段是否写回 CP0 寄存器
input	wire	5	wb_cp0_reg_write_addr	mem_wb	旁路信号, WB 阶段写回 CP0 寄存器地址
input	wire	32	wb_cp0_reg_data	mem_wb	旁路信号, WB 阶段写回 CP0 寄存器数据
input	wire	32	cp0_reg_data_i	cp0_reg	CP0 协处理器寄存器读出数据
input	wire	32	excepttype_i	id_ex	异常类型
input	wire	32	current_inst_address_i	id_ex	指令地址
output	reg	5	wd_o	ex_mem	通用寄存器写端口地址
output	reg	1	wreg_o	ex_mem	通用寄存器写端口使能
output	reg	32	wdata_o	ex_mem	通用寄存器写端口数据
output	wire	32	inst_o	ex_mem	指令内容
output	reg	32	hi_o	ex_mem	HI 寄存器写入数据
output	reg	32	lo_o	ex_mem	LO 寄存器写入数据
output	reg	1	whilo_o	ex_mem	HILO 寄存器写使能
output	wire	8	aluop_o	ex_mem	ALU 运算类型
output	wire	32	mem_addr_o	ex_mem	访存类型之类的访存地址
output	wire	32	reg2_o	ex_mem	ALU 第二个操作数
output	wire	1	stallreq	ctrl	EX 阶段的暂停请求
output	reg	5	cp0_reg_read_addr_o	cp0_reg	CP0 协处理器寄存器读出地址
output	reg	1	cp0_reg_we_o	ex_mem	CP0 协处理器寄存器写使能
output	reg	5	cp0_reg_write_addr_o	ex_mem	CP0 协处理器寄存器写入地址
output	reg	32	cp0_reg_data_o	ex_mem	CP0 协处理器寄存器写入数据
output	wire	32	excepttype_o	ex_mem	异常类型
output	wire	32	current_inst_address_o	ex_mem	指令地址
output	wire	1	is_in_delay_slot_o	ex_mem	当前指令是否位于分支延迟槽中

2.5.3 设计细节

EX 是指令流水阶段里的执行阶段，本阶段主要根据 ID 阶段传来的指令信息和使能信号进行相应的运算。运算主要分为两种，对于运算操作的运算和对于访存操作的运算。

对于运算操作的运算即根据 ID 给的操作数和操作标识信息进行相应的逻辑运算或算数运算。其中，逻辑运算主要指 AND、OR、XOR、NOT、NOR ; 算数运算主要指 ADD、SUB、MUL。由于本 CPU 面向 ucore 进行设计，没有支持除法运算。正常情况下，大部分的运算操作的运算结果在流水线上继续传递，用于在 WB 的时候回写。

对于访存操作的运算即根据 ID 给的操作分类信号，判断是否是访存指令。对于访存指令，由于 MIPS 32 支持的访存类型主要是基于基地址和地址偏移的，所以需要在本阶段计算出需要访存的具体地址（注意是虚地址），进而传递给 MEM 阶段进行正确的访存处理。

除了运算之外，本阶段与之前的阶段类似，也传递异常信息归总和暂停请求信号。本阶段主要用于发现运算溢出异常和陷入异常。

2.6 ex_mem

2.6.1 简介概述

ex_mem 用于衔接五级流水线第三阶段 ex 和第四阶段 mem，在时钟上升沿储存 ex 阶段数据，传递给 mem 阶段，将 ex 阶段需要写入的数据传递至 mem 阶段进行访存操作，是一个简单的时序逻辑电路。

2.6.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	时钟信号
input	wire	1	rst	CPU 外部	复位信号
input	wire	5	ex_wd	ex	通用寄存器写端口地址
input	wire	1	ex_wreg	ex	通用寄存器写端口使能
input	wire	32	ex_wdata	ex	通用寄存器写端口数据
input	wire	32	ex_hi	ex	HI 寄存器写入数据
input	wire	32	ex_lo	ex	LO 寄存器写入数据
input	wire	1	ex_whilo	ex	HILO 寄存器写使能
input	wire	8	ex_aluop	ex	ALU 运算类型
input	wire	32	ex_mem_addr	ex	访存指令的访存地址
input	wire	32	ex_reg2	ex	ALU 第二个操作数
input	wire	1	ex_cp0_reg_we	ex	CP0 协处理器寄存器写使能
input	wire	5	ex_cp0_reg_write_addr	ex	CP0 协处理器寄存器写入地址

input	wire	32	ex_cp0_reg_data	ex	CP0 协处理器寄存器写入数据
input	wire	32	ex_excepttype	ex	异常类型
input	wire	1	ex_is_in_delay_slot	ex	当前指令是否位于分支延迟槽中
input	wire	32	ex_current_inst_address	ex	指令地址
input	wire	32	ex_inst	ex	指令内容
input	wire	6	stall	ctrl	流水线暂停使能
input	wire	1	flush	ctrl	流水线清空使能
output	reg	5	mem_wd	mem	通用寄存器写端口地址
output	reg	1	mem_wreg	mem	通用寄存器写端口使能
output	reg	32	mem_wdata	mem	通用寄存器写端口数据
output	reg	32	mem_hi	mem	HI 寄存器写入数据
output	reg	32	mem_lo	mem	LO 寄存器写入数据
output	reg	1	mem_whilo	mem	HILO 寄存器写使能
output	reg	8	mem_aluop	mem	ALU 运算类型
output	reg	32	mem_mem_addr	mem	
output	reg	32	mem_reg2	mem	ALU 第二个操作数
output	reg	1	mem_cp0_reg_we	mem	CP0 协处理器寄存器写使能
output	reg	5	mem_cp0_reg_write_addr	mem	CP0 协处理器寄存器写入地址
output	reg	32	mem_cp0_reg_data	mem	CP0 协处理器寄存器写入数据
output	reg	32	mem_excepttype	mem	异常类型
output	reg	1	mem_is_in_delay_slot	mem	当前指令是否位于分支延迟槽中
output	reg	32	mem_current_inst_address	mem	指令地址
output	reg	32	mem_inst	mem	指令内容

2.6.3 设计细节

EX/MEM 是流水线寄存器，主要用于缓存 EX 阶段需要向别的阶段传递的操作信息和控制信息，并且在下一个时钟跳变的时候将这些信息分配给对应的处理模块。与之前的流水线寄存器类似，大部分情况下这些缓存的流水线信息都是流水传入 MEM 阶段。

同时，本阶段也与之前的流水线寄存器一样进行暂停和清空流水线处理。通过对于暂停信号的判断，判别是否需要暂停缓存 EX 阶段产生的信息并向 MEM 阶段传入空泡；通过清空流水线势能信号判断是否需要清空流水线，即清除缓存并传递空泡。

2.7 mem

2.7.1 简介概述

mem 阶段实现访存操作，是五级流水线的第四级，是一个复杂的组合逻辑电路。

2.7.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	rst	CPU 外部	复位信号
input	wire	5	wd_i	ex_mem	通用寄存器写端口地址
input	wire	1	wreg_i	ex_mem	通用寄存器写端口使能
input	wire	32	wdata_i	ex_mem	通用寄存器写端口数据
input	wire	32	hi_i	ex_mem	HI 寄存器写入数据
input	wire	32	lo_i	ex_mem	LO 寄存器写入数据
input	wire	1	whilo_i	ex_mem	HILO 寄存器写使能
input	wire	8	aluop_i	ex_mem	ALU 运算类型
input	wire	32	mem_addr_i	ex_mem	访存指令的访存地址
input	wire	32	reg2_i	ex_mem	ALU 第二个操作数
input	wire	32	mem_data_i	CPU 外部	Load 指令访存的结果
input	wire	1	tlb_hit	tlb_reg	TLB 是否命中
input	wire	32	physical_addr	tlb_reg	物理地址
input	wire	1	cp0_reg_we_i	ex_mem	CP0 协处理器寄存器写使能
input	wire	5	cp0_reg_write_addr_i	ex_mem	CP0 协处理器寄存器写入地址
input	wire	32	cp0_reg_data_i	ex_mem	CP0 协处理器寄存器写入数据
input	wire	32	excepttype_i	ex_mem	异常类型
input	wire	1	is_in_delay_slot_i	ex_mem	当前指令是否位于分支延迟槽中
input	wire	32	current_inst_address_i	ex_mem	指令地址
input	wire	32	cp0_status_i	cp0_reg	CP0 协处理器 Status 寄存器读出数据
input	wire	32	cp0_cause_i	cp0_reg	CP0 协处理器 Cause 寄存器读出数据
input	wire	32	cp0_epc_i	cp0_reg	CP0 协处理器 Epc 寄存器读出数据
input	wire	1	wb_cp0_reg_we	mem_wb	旁路信号, WB 阶段是否写回 CP0 寄存器

input	wire	5	wb_cp0_reg_write_addr	mem_wb	旁路信号，WB 阶段写回 CP0 寄存器地址
input	wire	32	wb_cp0_reg_data	mem_wb	旁路信号，WB 阶段写回 CP0 寄存器数据
input	wire	32	inst_i	ex_mem	指令内容
output	reg	5	wd_o	mem_wb	通用寄存器写端口地址
output	reg	1	wreg_o	mem_wb	通用寄存器写端口使能
output	reg	32	wdata_o	mem_wb	通用寄存器写端口数据
output	reg	32	hi_o	mem_wb	HI 寄存器写入数据
output	reg	32	lo_o	mem_wb	LO 寄存器写入数据
output	reg	1	whilo_o	mem_wb	HILO 寄存器写使能
output	reg	32	mem_addr_o	CPU 外部	外设写地址
output	wire	1	mem_we_o	CPU 外部	外设写使能
output	reg	4	mem_sel_o	CPU 外部	外设写片选
output	reg	32	mem_data_o	CPU 外部	外设写数据
output	reg	1	mem_ce_o	CPU 外部	外设写使能
output	wire	32	virtual_addr	tlb_reg	虚拟地址
output	reg	1	cp0_reg_we_o	mem_wb	CP0 协处理器寄存器写使能
output	reg	5	cp0_reg_write_addr_o	mem_wb	CP0 协处理器寄存器写入地址
output	reg	32	cp0_reg_data_o	mem_wb	CP0 协处理器寄存器写入数据
output	reg	32	excepttype_o	mem_wb	异常类型
output	wire	32	cp0_epc_o	ctrl	CP0 协处理器 Epc 寄存器数据
output	wire	1	is_in_delay_slot_o	cp0_reg	当前指令是否位于分支延迟槽中
output	wire	32	current_inst_address_o	cp0_reg	指令地址
output	reg	32	bad_address	cp0_reg	引发 TLB 缺失的虚址
output	wire	32	inst_o	mem_wb	指令内容

2.7.3 设计细节

MEM 阶段即访存阶段，本阶段的主要工作是识别访存指令，并根据相应的访存指令产生相应的片选信号通过 MMU 与外设或者存储设备进行交互。换句话说，本阶段的主要任务是获取访存指令所想要访问的存储器或外设的数据和向访存指令所想要访问的存储器或外设写入数据。值得一提的是，从 EX 阶段传递过来的指令地址是虚拟地址，需要进行虚实转化。所以访存的具体流程是：根据访存指令的 L/S 特性生成读或写势能；将需要访存的虚拟地址传入 MMU 进行虚实转化；将转化后的地址通过 MMU 的使能传入正确的存储设备或外设；读取或写入相应的数据。对于 L 类型的指令，获取的数据

会传入 WB 阶段进而写入目标寄存器。

与之前的几个阶段不同，本阶段不在传递异常信息归总。主要原因是，到 MEM 阶段以后，异常信息归总已经收集到了各个阶段发送的异常信息，并且发生异常的指令也不能进入 WB 阶段，所以在 MEM 阶段就要根据之前收集的异常信息进行异常处理。具体来说，即根据流水传递过来的异常信息归总，生成相应的异常类型标志信息，传递给协处理器 CP0 和控制器。应当注意的是，MEM 阶段也会识别出 TLB 缺失异常。

由于 MIPS 32 不会再访存阶段进行暂停请求，所以本阶段不再产生暂停请求标识。另外，对于上表中的 CPU 外部，实际上是先将信号接入 openmips 模块再向外连接的，不过出于理解和表述的方便，这里直接写为 CPU 外部，之后的类似情形也大抵如此。有关 openmips 综合模块，会在后文详细说明。

2.8 mem_wb

2.8.1 简介概述

mem_wb 用于衔接五级流水线第四阶段 mem 和第五阶段 wb，在时钟上升沿储存 mem 阶段数据，传递给对应的寄存器，是一个简单的时序逻辑电路。

2.8.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	时钟信号
input	wire	1	rst	CPU 外部	复位信号
input	wire	5	mem_wd	mem	通用寄存器写端口地址
input	wire	1	mem_wreg	mem	通用寄存器写端口使能
input	wire	32	mem_wdata	mem	通用寄存器写端口数据
input	wire	32	mem_hi	mem	HI 寄存器写入数据
input	wire	32	mem_lo	mem	LO 寄存器写入数据
input	wire	1	mem_whilo	mem	HILO 寄存器写使能
input	wire	1	mem_cp0_reg_we	mem	CP0 协处理器寄存器写使能
input	wire	5	mem_cp0_reg_write_addr	mem	CP0 协处理器寄存器写入地址
input	wire	32	mem_cp0_reg_data	mem	CP0 协处理器寄存器写入数据
input	wire	32	mem_inst	mem	指令内容
input	wire	6	stall	ctrl	流水线暂停使能
input	wire	1	flush	ctrl	流水线清空使能
output	reg	5	wb_wd	regfile	通用寄存器写端口地址
output	reg	1	wb_wreg	regfile	通用寄存器写端口使能
output	reg	32	wb_wdata	regfile	通用寄存器写端口数据

output	reg	32	wb_hi	hilo_reg	HI 寄存器写入数据
output	reg	32	wb_lo	hilo_reg	LO 寄存器写入数据
output	reg	1	wb_whilo	hilo_reg	HILO 寄存器写使能
output	reg	1	wb_cp0_reg_we	cp0_reg	CP0 协处理器寄存器写使能
output	reg	5	wb_cp0_reg_write_addr	cp0_reg	CP0 协处理器寄存器写入地址
output	reg	32	wb_cp0_reg_data	cp0_reg	CP0 协处理器寄存器写入数据
output	reg	32	wb_inst	tlb_reg	传递 TLBWI 和 TLBWR 指令

2.8.3 设计细节

本部分是介于 MEM 和 WB 阶段之间的流水线寄存器，主要用于缓存来自 MEM 阶段的控制信息、数据信息和使能信息，并在下一个时钟跳变把这些信息分配到所需要的各个模块。其中，有关于寄存器写入的信息和、关于 HILO 寄存器写入的信息、关于协处理写入的信息。换句话说，本模块主要用于控制寄存器写入有关信息的传递。

与之前的流水线寄存器类似，MEM/WB 也对控制器传递过来的暂停请求和清空流水请求作出会应。对于暂停请求，即缓存 MEM 阶段流入的信息，往 WB 阶段插入空泡；对于清空流水请求，即清空缓存寄存器里的信息，并传递空指令。与之前流水线寄存器不同的地方是，本流水线寄存器还往 tlb_reg 模块传递指令信息，主要是为了 tlb_reg 能正确的接收到 TLBWI 和 TLBWR 指令并处理，放在流水线的最后可以保证不会发生有关协处理器的数据、结构冲突。

2.9 regfile

2.9.1 简介概述

通用寄存器模块，主要负责 MIPS 32 的 32 个通用寄存器的维护，主要功能是存储、修改和输出。

2.9.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	时钟信号
input	wire	1	rst	CPU 外部	复位信号
input	wire	1	we	mem_wb	通用寄存器写端口使能
input	wire	5	waddr	mem_wb	通用寄存器写端口地址
input	wire	32	wdata	mem_wb	通用寄存器写端口数据
input	wire	1	re1	id	通用寄存器读端口 1 使能
input	wire	5	raddr1	id	通用寄存器读端口 1 地址

input	wire	1	re2	id	通用寄存器读端口 1 使能
input	wire	5	raddr2	id	通用寄存器读端口 1 地址
output	reg	32	rdata1	id	通用寄存器读端口 1 数据
output	reg	32	rdata2	id	通用寄存器读端口 1 数据

2.9.3 设计细节

本模块是通过硬件描述语言实现的 32 个 MIPS 32 标准下的通用寄存器。寄存器即电路中的锁存器，除了存储相应的通用寄存器信息外，本模块有两个主要的功能：根据使能信号和数据信息修改通用寄存器的内容；根据使能信号输出通用寄存器的内容。

对于前者，本模块通过识别输入的使能信息判断是否是写操作，如果是则将传递入的数据写入到相应的通用寄存器中；对于后者，则是根据读使能和读的地址将相应寄存器里的内容输出。值得一提的是，由于 MIPS 32 的指令语句里会有同时将两个通用寄存器作为运算数的情况，所以需要设计两个输出通用寄存器的线路，他们是并行且平行的。

2.10 hilo_reg

2.10.1 简介概述

用于支持 MIPS 32 乘法运算的 HI/LO 寄存器。

2.10.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	时钟信号
input	wire	1	rst	CPU 外部	复位信号
input	wire	1	we	mem_wb	HILO 寄存器写使能
input	wire	32	hi_i	mem_wb	HI 寄存器写入数据
input	wire	32	lo_i	mem_wb	LO 寄存器写入数据
output	reg	32	hi_o	ex	HI 寄存器读出数据
output	reg	32	lo_o	ex	LO 寄存器读出数据

2.10.3 设计细节

本模块即 MIPS 32 乘法运算中的 HI/LO 寄存器，与通用寄存器 regfile 模块类似，此模块主要用于维护 HI 寄存器和 LO 寄存器，并对其进行读和写。根据相应的使能，本模块会将传递来的写入数据写入到 HI/LO 寄存器中。同时，本模块会一直输出 HI/LO 寄存器里存储的数据，用于相应的指令在五级流水中使用。

Chapter 3

控制模块

上文提到，指令流水 CPU 的设计中除了运算部分另外一个重要的部分是控制部分，本章节主要介绍我们设计的 CPU 中的控制模块相关的部分。所谓的控制部分就是 CPU 中用于控制各条指令执行的模块，其主要功能是暂停和清空。所谓暂停是指在 CPU 执行的过程中，可能由于各类冲突所需要的插空泡操作；所谓清空是指在 CPU 执行过程中，跳转的时候需要清空还未执行的还在流水线上的指令。

另外，MIPS 32 是基于异常的一类指令架构。异常即是 CPU 在处理指令的过程中可能发生的需要 OS 介入的情况，异常处理将 CPU 分为了用户态和内核态，正常情况下，CPU 是运行在用户态的，而当异常发生的时候，CPU 就需要进入内核态让 OS 介入进行异常的处理。为了让 CPU 介入异常的处理，除了使控制模块正确的跳转到异常处理程序的入口外还需要对异常的一些基本信息进行存储，这就涉及到了协处理器 CP0，所以本章节主要介绍控制模块和协处理器 CP0 设计。

3.1 ctrl

3.1.1 简介概述

控制器模块，主要用于暂停和清空两类控制信号的生成和异常部分跳转地址的控制。

3.1.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	CPU 工作时钟
input	wire	1	rst	CPU 外部	异步清零信号
input	wire	1	stallreq_from_id	id	id 阶段要求暂停
input	wire	1	stallreq_from_ex	ex	ex 阶段要求暂停
input	wire	1	stallreq_from_mem	mem	mem 阶段要求暂停
input	wire	1	mem_we_i	mem	mem 阶段写使能信号
input	wire	32	ebase_i	cp0_reg	异常处理向量
input	wire	32	excepttype_i	mem	异常标识

input	wire	32	cp0_epc_i	mem	经过数据冲突处理的 cp0_epc 信息
output	reg	6	stall	各个流水线寄存器阶段	暂停控制信息
output	reg	1	mem_we_o	CPU 外部	外设写使能
output	reg	32	new_pc	pc_reg	发送异常的下一条指令地址
output	reg	1	flush	各个流水线寄存器阶段	清空流水线控制信息

3.1.3 设计细节

本模块是我们设计 CPU 的控制模块，值得一提的是，由于我们写的是流水线 CPU，所以实际上控制信息已经散落在了流水线的各个阶段，相当于控制信息也进行了流水。所以该模块的主要功能虽然表面上是控制信号的生成，实际上可以认为是控制信息的总结、仲裁、生成和转发。

由于流水线 CPU 一个时钟周期里会有多条指令在不同的阶段并行执行，所以从各个阶段都有可能发来暂停的请求。具体来说，这些暂停请求可能来自 id、ex、mem 三个阶段。对于这三个阶段发来的暂停请求，基于 MIPS 32 标准，我们处理的暂停请求优先级是 ex、id、mem。对于暂停请求的处理即生成相应的暂停控制向量输出到各级流水阶段衔接的流水寄存器中。另外，由于处理的特殊性，这里特别说明下 mem 阶段的暂停请求。mem 阶段的暂停请求主要是为了解决结构冲突而执行的，即对于 L 型指令我们要保证他已经会写到寄存器中了，才能对后面的指令进行译码。但是，由于我们的计算机是异步的计算机，意思是计算机中 CPU 的时钟和外设的时钟不是同步时钟。所以我们就要保证，当我们的外设写入使能端使能的时候，数据应该已经准备好了，不然就会将错误的数据写入外设。所以我们将 mem 的暂停分为了两状态的状态机，第一个状态让包括 mem 阶段的流水暂停，而不给出写使能信号；第二个状态让 mem 开始流水，并给出写使能信号。这样，就相当于我们使用了一个 CPU 周期的时间让 MEM 阶段的写入数据收敛到一个稳定的状态，避免之前说的错误的发生。

除了各个阶段发来的暂停请求以外，该模块的另外一个重要的处理是对于异常发生时候的处理。根究 mem 发送来的异常信息，ctrl 模块可以识别出发送了哪一种异常，进而告知并控制 pc_reg 对于下一条指令的取指地址，即异常处理程序的入口地址。而当异常发生的时候，本模块还需要对各个流水线寄存器模块发送 flush 信号，即流水清空信号，清空目前的流水线。这么做的原因是，引发异常的指令之后的指令都不应该执行完毕，不能让这些指令进入 WB 阶段，所以要清空各个流水线寄存器，让流水线看起来根本没有工作。这也是 MIPS 精确异常的要求。

3.2 cp0_reg

3.2.1 简介概述

协处理器 CP0 模块，主要实现了有关 uCore 的协处理器 CP0 相关寄存器、有关寄存器的读写和异常发生时相关寄存器的设置。

3.2.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	CPU 工作时钟
input	wire	1	rst	CPU 外部	异步清零信号
input	wire	1	we_i	mem_wb	CP0 写使能
input	wire	5	waddr_i	mem_wb	CP0 写地址
input	wire	5	raddr_i	ex	CP0 读地址
input	wire	32	data_i	mem_wb	CP0 写数据
input	wire	6	int_i	CPU 外部	CPU 外部中断信号
input	wire	32	bad_address_i	mem	发生 TLB 缺失的虚地址
input	wire	32	excepttype_i	mem	异常标识
input	wire	32	current_inst_addr_i	mem	目前指令的虚地址
input	wire	1	is_in_delay_slot_i	mem	标记当前指令是否在延迟槽中
output	reg	32	data_o	ex	输出指令所读的 CP0 寄存器
output	reg	32	count_o	无指定	输出 CP0_count 寄存器
output	reg	32	compare_o	无指定	输出 CP0_compare 寄存器
output	reg	32	status_o	无指定	输出 CP0_status 寄存器
output	reg	32	cause_o	无指定	输出 CP0_cause 寄存器
output	reg	32	epc_o	无指定	输出 CP0_epc 寄存器
output	reg	32	config_o	无指定	输出 CP0_config 寄存器
output	reg	32	ebase_o	无指定	输出 CP0_ebase 寄存器
output	reg	32	index_o,	无指定	输出 CP0_index 寄存器
output	reg	32	random_o,	无指定	输出 CP0_random 寄存器
output	reg	32	entrylo0_o,	无指定	输出 CP0_entrylo0 寄存器
output	reg	32	entrylo1_o,	无指定	输出 CP0_entrylo1 寄存器
output	reg	32	pagemask_o,	无指定	输出 CP0_pagemask 寄存器
output	reg	32	badvaddr_o,	无指定	输出 CP0_badvaddr 寄存器
output	reg	32	entryhi_o,	无指定	输出 CP0_entryhi 寄存器
output	reg	1	timer_int_o,	CPU 外部	时钟中断（可反接）

3.2.3 设计细节

本模块主要是对 MIPS 32 标准中的协处理器 CP0 进行实现，主要实现与 uCore 有关的 CP0 寄存器及在这些寄存器上的有关操作。协处理器，和它的字面意思一样，在 MIPS 中是为了协助 CPU 在硬

件层面处理有关问题而设置的特殊寄存器。它与通用寄存器不同，一般情况下，是不能对协处理器 CP0 中的寄存器进行读写的。协处理器主要的作用是在硬件层面保存与 CPU 处理事务有关的信息，进而成为处理器硬件和操作系统软件交互的桥梁。

我们实现的协处理器 CP0 是面向 uCore 设计的，并没有实现 MIPS 32 文档中所有的 CP0 寄存器。具体来说，我们实现了下列 CP0 寄存器：data、count、compare、status、cause、epc、config、ebase、index、random、entrylo0、entrylo1、pagemask、badvaddr、entryhi，共记 15 个寄存器。其中，前 8 个是为了处理常规异常设置的寄存器，而后 7 个是为了处理 TLB 异常设置的寄存器。这些寄存器实现的格式都与 MIPS 32 文档完全一致。

对于 CP0 寄存器的读写操作，与通用寄存器 regfile 模块类似，也是通过传递使能信号和数据信息进行的。与通用寄存器模块不同的是，CP0 寄存器的读和写有一定的 MIPS 规范，比如有一些寄存器只读、有一些寄存器可写，再比如有一些寄存器只有一些字段可写，这些均按照 MIPS 32 规范设计。另外，CP0 在处理异常中也有很大的作用，他需要根据触发异常的指令再硬件层面存储一些有关异常的信息（发生了什么异常、哪里发生异常、TLB 缺失的虚地址是等）。设计层面上，即把流水阶段我们得到的异常综合信息和相关的地址和数据信息连入 CP0 模块，进而进行锁存器的赋值即可。

值得一提的是，CP0 模块除了常规的处理 MTC0 和 MFC0 的读写方式外，还稳定的通过电路向外输出可读存储器的信息，即上表中所指的无指定的输出单元。这里的无指定不代表这些输出单元没有被键入某个特定的模块，而是指它衔接到需要这些 CP0 寄存器信息的单元，没有特定的单元。换句话说，他们是按需分配的。

Chapter 4

内存管理

对于 MIPS 32 来说，虚拟内存的概念是十分重要的。具体来说，虚拟内存指用户或者 OS 在编写软件的过程中使用的地址是固定大小的虚拟内存空间。这些虚拟的内存空间地址不一一对应到实际的地址空间，而是通过虚实转化的方式来在硬件层面得到实际操作对应的物理地址。这样设计的好处是明显的：一是程序或软件开发人员不用涉及物理地址的分配问题，将这一过程统一交给 OS 来处理；二是可以动态的管理每个虚拟内存地址对应的实际地址，使得实际地址能被有效的利用。

因为 CPU 在运行的过程中也需要虚实地址的转换，所以在硬件层面我们也需要虚实地址的转换元件。为了在硬件层面实现虚实转换，常用的办法是加入快表 TLB。其工作过程即对于地址虚实转化先查询快表，对于可以查询到的虚实对应关系直接进行转化，不需要软件 OS 的介入；对于查询不到的，通过 TLB 异常的形式告知 OS，让 OS 查询内存中大的页表后，设置相应的 TLB 表项实现虚实转化。值得一提的是，TLB 由于是硬件层面的 cache 结构，可以大大加速虚拟地址到物理地址的转化过程。

除了虚拟地址和实际地址通过页表的转化过程，实际上在 MIPS 规范中，已经将虚拟地址按块划分，并非所有的虚拟地址都需要页表转化。具体来说，MIPS 32 标准中，将 32 位的地址空间（共计 4GB）分为四个部分：kuseg（低 2GB），用户空间，需要虚实转化；kseg0（加 512MB），将第一位置 0 即为实际地址；kseg1（加 512MB），将前三位置 0 即为实际地址；kseg2（高 1GB），内核空间，需要虚实转化。

在设计过程中，uCore 将不同的设备放置在了不同的地址空间。

段（权限）	**虚拟地址**	**映射目标（物理地址）**	**备注**
kuseg（用户态）	0x00000000 - 0x7FFFFFFF	RAM（通过查询 TLB 动态确定）	用户程序
kseg0（内核态）	0x80000000 - 0x807FFFFF	RAM（0x00000000 - 0x007FFFFF）	开机时存放操作系统
kseg1（内核态）	0xBE000000 - 0xBEFFFFFF	FLASH	关机时存放操作系统
	0xBFC00000 - 0xBFC00FFF	ROM	存放 BootLoader
	0xBFD003F8 - 0xBFD003FC	串口	串口数据/串口状态
	0xBA000000 - 0xBA080000	VGA	显存，用于显示图像
kseg2（内核态）	0xC0000000 - 0xFFFFFFFF	RAM（通过查询 TLB 动态确定）	

4.1 tlb_reg

4.1.1 简介概述

快表模块，实现了 TLB 表，相当于一个数据选择器。除了 TLB 表查询外，还肩负非查 TLB 表的虚实转化和外设使能控制的功能，可以看出是一个内存管理单元 MMU。

4.1.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	CPU 工作时钟
input	wire	1	rst	CPU 外部	异步清零信号
input	wire	32	addr_i	pc_reg 或 mem	需要转化的虚地址
input	wire	32	inst_i	mem/wb	wb 阶段的指令
input	wire	32	index_i	cp0	CP0_index 输入
input	wire	32	random_i	cp0	CP0_random 输入
input	wire	32	entrylo0_i	cp0	CP0_entrylo0 输入
input	wire	32	entrylo1_i	cp0	CP0_entrylo1 输入
input	wire	32	entryhi_i	cp0	CP0_entryhi 输入
output	reg	1	tlb_hit,	pc_reg 或 mem	转化是否成功标记
output	reg	1	sram_ce	CPU 外部	sram 使能信号
output	reg	1	flash_ce	CPU 外部	flash 使能信号
output	reg	1	rom_ce	CPU 外部	rom 使能信号
output	reg	1	serial_ce	CPU 外部	串口使能信号
output	reg	1	vga_ce	CPU 外部	vga 输出使能
output	reg	32	addr_o	pc_reg 或 mem	转化出的物理地址

4.1.3 设计细节

本模块的主体是 TLB 快表，实现了一个有 16 个表项的 MIPS 32 标准快表，这也是为什么我们将之命名为 tlb_reg。但同时，本模块中还实现了对于不需要虚实转化地址的转化工作，即 kseg0 和 kseg1 段的地址，并且根据 uCore 对于虚拟地址的分配实现了对于外设使能端的控制。所以，总的来说，可以把这个模块看成是一个内存管理子单元，其与后面提到的 openmips 模块共同实现关于内存的管理。

对于 TLB 快表的实现，主要是关于查表和设表两个部分。查表是有关 MIPS 32 标准的实现，在这里就不再具体展开说明其设计。对于设表，通过 uCore 的内容，我们知道主要指的是 TLBWI 和 TLBWR 两个指令。TLBWI 即根据 index 进行 tlb 表项的设置哦，TLBWR 即根据 random 进行 tlb 表项的设置，所以我们需要从 CP0 模块引入相应的寄存器内容。

对于 kseg0、kseg1 和外设使能的实现，仅需要判断输入的虚拟地址是否落在这两个区间内，若落在这两个区间内是否落在相应的外设配置地址即可。

很明显，在 if 和 mem 阶段我们都涉及到了虚实转化的问题。理论上，一个可行的解决方式是，将

这两个部分的虚实转化接入一个控制模块，然后通过这个控制模块对这两个虚实转化哪个使用 tlb_reg 进行取舍。但是，在我们设计的 CPU 中，我们采用两个完全一致的 tlb_reg 分别接入 if 和 mem 阶段来解决这个问题。这种解决方法的不同在于还需要修改 ctrl 模块去适应两个 tlb_reg 可能发生的冲突 (比如 if、mem 同时需要对同一个外设读取)。注意，这两个 tlb_reg 是完全一致，同时，tlb_miss 标志是否使能是 TLB 缺失异常是否发生的充分必要条件。

4.2 openmips

4.2.1 简介概述

综合模块，用于连接 CPU 内部的诸多模块，并且把外部有关的信号输入 CPU 的特定模块或把 CPU 特定模块的信号输出到外部。

4.2.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	CPU 外部	CPU 工作时钟
input	wire	1	rst	CPU 外部	异步清零信号
input	wire	6	int_i	CPU 外部	中断使能
input	wire	32	if_data_i	CPU 外部	if 阶段取到的指令
input	wire	32	mem_data_i	CPU 外部	mem 阶段取到的数据
output	wire	32	if_addr_o	外设	if 阶段所取指令的物理地址
output	wire	1	if_sram_ce_o	sram	if 阶段 sram 使能信号
output	wire	1	if_flash_ce_o	flash	if 阶段 flash 使能信号
output	wire	1	if_rom_ce_o	rom	if 阶段 rom 使能信号
output	wire	1	if_serial_ce_o	串口	if 阶段串口使能信号
output	wire	1	if_vga_ce_o	vga	if 阶段 vga 使能信号
output	wire	32	mem_addr_o	外设	mem 阶段访存的物理地址
output	wire	32	mem_data_o	外设	mem 阶段访存的输出数据
output	wire	1	mem_we_o	外设	mem 阶段访存的写使能
output	wire	4	mem_sel_o	外设	mem 阶段访存的片选信号
output	wire	1	mem_sram_ce_o	sram	mem 阶段 sram 使能信号
output	wire	1	mem_flash_ce_o	flash	mem 阶段 flash 使能信号
output	wire	1	mem_rom_ce_o	rom	mem 阶段 rom 使能信号
output	wire	1	mem_serial_ce_o	串口	mem 阶段串口使能信号
output	wire	1	mem_vga_ce_o	vga	mem 阶段 vga 使能信号
output	wire	1	mem_ce_o	外设	mem 阶段总使能信号
output	wire	1	timer_int_o	外设	时钟中断信号

4.2.3 设计细节

本部分是对于 CPU 各个模块的综合，即实例化之前提过的各个模块类，并通过 wire 衔接各个不同的实例元件对应的接口。同时，本部分将各个模块需要连到 CPU 外部（包括输入和输出）的部分通过 wire 连接到这一总模块的接口上，实现与 CPU 外部的交互。换句话说，本模块实现的是一个完整的基于 MIPS 32 的面向 uCore 设计的 CPU 模块类。

Chapter 5

外设连接

CPU 是计算机的核心所在，但是只有有了外设，才能称之为一台完整的计算机。在本节中，将介绍 ROM、RAM、Flash、串口、VGA、七段数码管、LED 灯、开关等外设的作用与设计（或使用方法）。

5.1 ROM

5.1.1 简介概述

ROM 即为 Read Only Memory，只读内存区，仅用于存储 Boot Loader。Boot Loader 是一段代码，CPU 执行这段代码在系统启动初期将 Flash 中的 uCore 操作系统数据拷贝至 RAM 中。

5.1.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	clk	thinpad_top	时钟信号
input	wire	1	ce	thinpad_top	使能信号
input	wire	12	addr	thinpad_top	读入地址
output	reg	32	inst	thinpad_top	读出指令

5.1.3 设计细节

Thinpad 开发板并未集成 ROM，ROM 采用 Verilog 语言实现。由于 Boot Loader 代码较短，直接用 case 语句实现。在时钟的上升沿，若 ROM 被使能，给定一个地址，返回一条指令。由于 ROM 使用 Verilog 语言在 FPGA 中实现，故可以忽略读延迟，无需考虑时序关系。

ROM 采用字编址，字长 32 位。

5.2 RAM

5.2.1 简介概述

RAM 即为 Random Access Memory，随机访问存储器，断电数据消失，系统启动后作为系统的主存，load/store 指令与此交互。

5.2.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	ce_n	thinpad_top	片选信号
input	wire	1	oe_n	thinpad_top	读使能信号
input	wire	1	we_n	thinpad_top	写使能地址
input	wire	4	be_n	thinpad_top	字节选择信号
input	wire	20	addr	thinpad_top	读写地址
inout	wire	32	data	thinpad_top	读写数据

注：n 表示 active low，低电平有效。

5.2.3 设计细节

RAM 集成在 Thinpad 开发板上，共有 2 块，分别为 Base RAM 和 Extern RAM，大小均为 4MB，采用字编址，地址线 20 位，数据线 32 位。

在使用 RAM 时，应阅读 RAM 的外设文档，理解清楚读写的时序关系，计算各个阶段的延迟。

读 RAM 时，置片选信号为 0，读使能信号为 0，写使能信号为 1，设置字节选择信号和地址，便可以读出数据。

写 RAM 时，置片选信号为 0，读使能信号为 X（任意态），写使能信号为 0，设置字节选择信号的地址，便可以写入数据。

在实现的过程中，有一条极为重要的时序约束：写 RAM 时，字节选择信号和地址必须早于写使能信号到达，必须晚于写使能信号撤离，否则会造成覆盖写入（本希望写入 1 字节数据，写入多个字节）或是错误写入（本希望写入地址 A，写入到地址 B）。

为了实现时序约束，在 CPU 内部采用状态机的方式实现，每条 store 指令都会被分为三个周期，第一个周期传递字节选择信号和地址，第二个周期拉低片选信号的写使能信号，第三个周期撤去字节选择信号和地址。

5.3 Flash

5.3.1 简介概述

Flash，快闪存储器，断电数据不消失，系统启动前存放 uCore 操作系统，仅在 Boot 阶段与此交互，从 Flash 中读出 uCore。

5.3.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	1	ce_n	thinpad_top	片选信号
input	wire	1	oe_n	thinpad_top	读使能信号
input	wire	1	we_n	thinpad_top	写使能地址
input	wire	4	byte_n	thinpad_top	字节模式信号
input	wire	1	rp_n	thinpad_top	工作模式信号
input	wire	1	vpen	thinpad_top	写保护
input	wire	23	a	thinpad_top	读写地址
inout	wire	16	data	thinpad_top	读写数据

注：n 表示 active low，低电平有效。

5.3.3 设计细节

Flash 集成在 Thinpad 开发板上，大小为 8MB，采用字节编址，地址线 23 位，数据线 16 位。

在使用 Flash 时，应阅读 Flash 的外设文档，理解清楚读写的时序关系，计算各个阶段的延迟。

读 Flash 时，置片选信号为 0，读使能信号为 0，写使能信号为 1，字节模式信号为 1（表示一次读出 16 个 bit），工作模式信号为 1（表示工作在正常模式），设置字节选择信号和地址，便可以读出数据。

读 Flash 前，要先向 Flash 写入 0xFF，将 Flash 转变为读模式。

5.4 串口

5.4.1 简介概述

串口，串行通信接口，用于计算机和外界进行通信。计算机内部均为并行信号，收发数据采用串行信号，串口进行并行数据与串行数据的转换，

5.4.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	8	TxD_data	thinpad_top	发数据
input	wire	1	TxD_busy	thinpad_top	发端口繁忙
input	wire	1	TxD_start	thinpad_top	发端口发送数据
output	wire	8	RxD_data	thinpad_top	收数据
output	wire	1	RxD_idle	thinpad_top	收端口空闲
output	wire	1	RxD_data_ready	thinpad_top	收端口就绪

5.4.3 设计细节

发送数据时，将 TxD_start 置为 1，TxD_data 传入数据。

接收数据时，利用 RxD_data_ready 触发 CPU 中断，调用中断处理程序，中断读数据。

应当注意的是，由于串口的时钟与 CPU 时钟的异步特性，需要加入一个缓存缓冲从串口接受的数据。

5.5 VGA

5.5.1 简介概述

VGA 即为 Video Graphics Array，视频图形阵列，用于向屏幕输出图像。

5.5.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	8	pixel	thinpad_top	像素数据
input	wire	1	hsync	thinpad_top	水平同步信号
input	wire	1	vsync	thinpad_top	垂直同步信号
input	wire	1	clk	thinpad_top	时钟信号
input	wire	1	de	thinpad_top	数据使能信号

5.5.3 设计细节

Thinpad 开发板上提供了 HDMI 接口，可以向屏幕输出图像，其显示像素位宽为 8，分别是 R（3bit）、G（3bit）、B（2bit），最终实现了 800x600 刷新率 75Hz 的图像显示功能，

VGA 输出的原理为逐行扫描，同时需要多扫描一段区域并加入一些同步信号，工程模板中提供了逐行扫描模块的实现代码，详见 VGA 的外设文档。

实现过程中需要一块显存。由于显示的原理为扫描，故需要一块区域存储向屏幕输出的数据，同时需要支持同时读写，给定一个地址读写一个 8 位宽的色彩像素。显存借助于 Vivado 提供的简单双端 Block Memory IP 核实现，读写数据宽度 8 位，读写深度 480000（800x600）。

在 MMU 中将 0xBA000000-0xBA080000 映射至 IP 核实现的显存位置，CPU 向这段地址写入即访问显存，同时另一读端口接入逐行扫描的模块，即可输出图像。

5.6 七段数码管

5.6.1 简介概述

七段数码管用于显示十进制数字，用于开发前期的调试工作。

5.6.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	16	leds	thinpad_top	数据

5.6.3 设计细节

Thinpad 开发板上集成了 2 个七段数码管，可以用来显示十进制数字，在调试初期非常有用，通过 MMU 将某一地址映射至七段数码管后可以用来显示指令内容、指令地址等等。

七段数码管需要提供一个模块将位宽为 4 的二进制信号转换为 0-F 的七段数码管表示，在提供的模板工程里实现了这份代码，实现原理也非常简单，实现一个 case 语句即可。

在功能测例里通过七段数码管中显示通过测例的数目，将功能测例里定义的七段数码管地址通过 MMU 实现映射，CPU 将数据写至该地址即实现了七段数码管的输出，从而可以清楚地显示测试情况。

5.7 LED 灯

5.7.1 简介概述

LED 灯用于显示二进制数字，用于开发前期的调试工作。

5.7.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述
input	wire	16	leds	thinpad_top	数据

5.7.3 设计细节

Thinpad 开发板上集成了 16 个 LED 灯，可以用来显示二进制数字，在调试初期非常有用，通过 MMU 将某一地址映射至 LED 灯后可以用来显示指令内容、指令地址等等。

将 leds 信号相应位置 1 该灯即点亮，原理非常简单。

5.8 开关

5.8.1 简介概述

开关用于提供简单的控制功能，用于开发前期的调试工作。

5.8.2 接口定义

信号类型	信号规格	信号位宽	信号名称	来源/去向	详细描述

output	wire	32	dip_sw	thinpad_top	32 个拨动开关
output	wire	6	touch_btn	thinpad_top	6 个按动开关

5.8.3 设计细节

Thinpad 开发板上集成了 32 个拨动开关和 6 个按动开关（其中 2 个为 clk 和 reset，已消除抖动，剩下 4 个需要自己实现消除抖动），用于开发前期提供最基础的控制功能。

开发的整个过程中，都需要 rst 开关提供复位信号，清零 CPU 状态。

开发初期为了 Debug，可以七段数码管输出指令地址后 8 位，LED 输出指令后 16 位，手按 clk 时钟进行调试。

同时可以将左边 32 个波动开关设置为 CPU 控制信号，例如左边某个开关为 1 时，CPU 进入调试模式，需要手按时钟。