重庆大学课程设计报告

课程设计题目:	MIPS SOC 设计与性能优化
学　　院:	计算机学院
专 业 班 级:	计算机科学与技术
年　　级:	2021
学　　生:	苟豪爽　　陈涛
	张婷婷　　邹奕
学　　号:	20215078　　20214307
	20214750　　20214415
完 成 时 间:	2024 年　01 月　15 日
成　　绩:	
指 导 教 师:	肖春华

重庆大学教务处制

项目	分值	优秀 $100 > x \geq 90$	良好 $90 > x \geq 70$	中等 $80 > x \geq 70$	及格 $70 > x \geq 60$	不及格 $x < 60$	评分
		参考标准					
学习态度	15	学习态度认真，科学作风严谨，严格保证设计时间并按任务书中规定的进度开展各项工作	学习态度比较认真，科学作风良好，能按期圆满完成任务书规定的任务	学习态度尚好，遵守组织纪律，基本保证设计时间，按期完成各项工作	学习态度尚可，能遵守组织纪律，能按期完成任务	学习马虎，纪律涣散，工作作风不严谨，不能保证设计时间和进度	
技术水平与实际能力	25	设计合理、理论分析与计算正确，实验数据准确，有很强的实际动手能力、经济分析能力和计算机应用能力，文献查阅能力强、引用合理、调查调研非常合理、可信	设计合理、理论分析与计算正确，实验数据比较准确，有较强的实际动手能力、经济分析能力和计算机应用能力，文献引用、调查调研比较合理、可信	设计合理，理论分析与计算基本正确，实验数据比较准确，有一定的实际动手能力，主要文献引用、调查调研比较可信	设计基本合理，理论分析与计算无大错，实验数据无大错	设计不合理，理论分析与计算有原则错误，实验数据不可靠，实际动手能力差，文献引用、调查调研有较大的问题	
创新	10	有重大改进或独特见解，有一定实用价值	有较大改进或新颖的见解，实用性尚可	有一定改进或新的见解	有一定见解	观念陈旧	
论文(计算书、图纸)撰写质量	50	结构严谨，逻辑性强，层次清晰，语言准确，文字流畅，完全符合规范化要求，书写工整或用计算机打印成文；图纸非常工整、清晰	结构合理，符合逻辑，文章层次分明，语言准确，文字流畅，符合规范化要求，书写工整或用计算机打印成文；图纸工整、清晰	结构合理，层次较为分明，文理通顺，基本达到规范化要求，书写比较工整；图纸比较工整、清晰	结构基本合理，逻辑基本清楚，文字尚通顺，勉强达到规范化要求；图纸比较工整	内容空泛，结构混乱，文字表达不清，错别字较多，达不到规范化要求；图纸不工整或不清晰	

指导教师评定成绩：

指导教师签名：

MIPS SOC 设计报告

苟豪爽、陈涛、张婷婷、邹奕

1 设计简介

我们小组在《计算机组成原理》实验 4 实现的简易五级流水线 CPU 的基础上，扩展实现了包括逻辑运算指令、移位指令、数据移动指令、算术运算指令、分支跳转指令、访存指令、内陷指令和特权指令，总计 57 条指令。成功连接了 Axi 接口，增加了 cache 成功测试通过了功能测试 89 个测试点，且上板测试也成功。

我们小组使用了 System Verilog 语言，对各模块可能产生的信号和数据等使用结构体进行封装，便于模块间的接口连接。我们将五级流水线不同阶段各为一个模块，使用流水线寄存器传输各模块的信号和数据，并通过数据通路对各模块和流水线寄存器进行连接。

1.1 小组分工说明

- 苟豪爽：负责 Cache 和接口实现，撰写实验报告；
- 陈涛：负责除法器、分支跳转和特权指令实现和扩展，撰写实验报告；
- 张婷婷：负责算术运算指令（除 DIV 部分）、访存指令和内陷指令的实现，撰写实验报告；
- 邹奕：负责逻辑运算，移位，数据移动指令实现和扩展，撰写实验报告；

2 设计方案

2.1 总体设计思路

CPU 是五级流水设计，每个阶段之间有相应的触发器来实现阶段之间信号量的传递，流水线由取指 (Fetch)，译码 (Decode)，执行 (Execute)，访存 (Memory)，写回 (WriteBack) 五个阶段组成。

mips 模块由 main_decode 和 datapath 模块组成，其中 main_decode 模块用于判断不同指令并进行译码操作，包括 alu 参与的指令的译码以及所有指令所对应译码以产生相应的控制信号。datapath 模块包括阶段取指 fetch，译码 decode，运算 execute，访存 memory，写回 write_back 5 个模块构成，同时在流水线文件 pipeline_reg 中包括他们的暂

存区：f_d_reg, d_e_reg, e_m_reg, m_w_reg 模块，以及包括全局控制流水线暂停的 ctrl 模块。

2.2 主要模块设计

2.2.1 cpu 模块设计

CPU 结构采用的是基础的五级流水线设计，流水线由取指 (Fetch)，译码 (Decode)，执行 (Execute)，访存 (Memory)，写回 (WriteBack) 五个阶段组成。CPU 由数据通路模块，缓存模块，控制模块等组成。

其中 Datapath 模块为五级流水的数据通路，主要完成了流水线取指，译码，执行，访存，写回功能。此外，在该模块中，实现了协处理器 CP0 以支持异常相关指令，控制模块 ctrl 以解决数据冲突，hilo 寄存器等；

其中 Cache 模块实现了对 CPU 与内存之间的缓存。

2.2.2 MIPS 模块设计

mips 模块接口说明：

Port name	Direction	Type	Description
clk	input	wire	时钟信号
rst	input	wire	复位信号
ext_int	input	[5:0]	硬件中断
inst_sram_addr	output	word_t	指令存储器的地址
inst_sram_rdata	input	word_t	从指令存储器读取的数据
memory_data_out	output	memory_data_out_t	从内存读取的数据
data_sram_rdata	input	word_t	从数据存储器读取的数据
debug_wb_pc	output	word_t	写回 PC 值
debug_wb_rf_wen	output	wen_t	寄存器写使能

Port name	Direction	Type	Description
debug_wb_rf_wnum	output	reg_addr_t	写回目的寄存器号
debug_wb_rf_wdata	output	word_t	写回存储器的数据
inst_stall	input		暂停指令执行
data_stall	input		暂停数据执行
flush	output		清空缓存

2.2.3 Datapath 模块

Datapath 模块接口说明:

Port name	Direction	Type	Description
clk	input	wire	
rst	input	wire	
inst_sram_data	input	inst_sram_data_t	取出的指令数据
fetch_data_out	output	fetch_data_out_t	取指阶段输出数据
data_sram_rdata	input	word_t	读到的数据存储器数据
memory_data_out	output	memory_data_out_t	访存阶段输出数据
wb_data_out	output	wb_data_out_t	写回阶段输出数据
execute_sign	output	control_sign_t	运算阶段信号
decode_data	output	decode_data_t	译码阶段数据
control_data	input	control_data_t	main_decode 模块译码阶段数据

Port name	Direction	Type	Description
clk	input	wire	
rst	input	wire	
decode_sign	input	control_sign_t	译码阶段信号
cache_flush	output	logic	cache 刷新信号
inst_stall	input	logic	关于 axi 接口的指令暂停信号
data_stall	input	logic	关于 axi 接口的数据暂停信号

Datapath 模块结构：

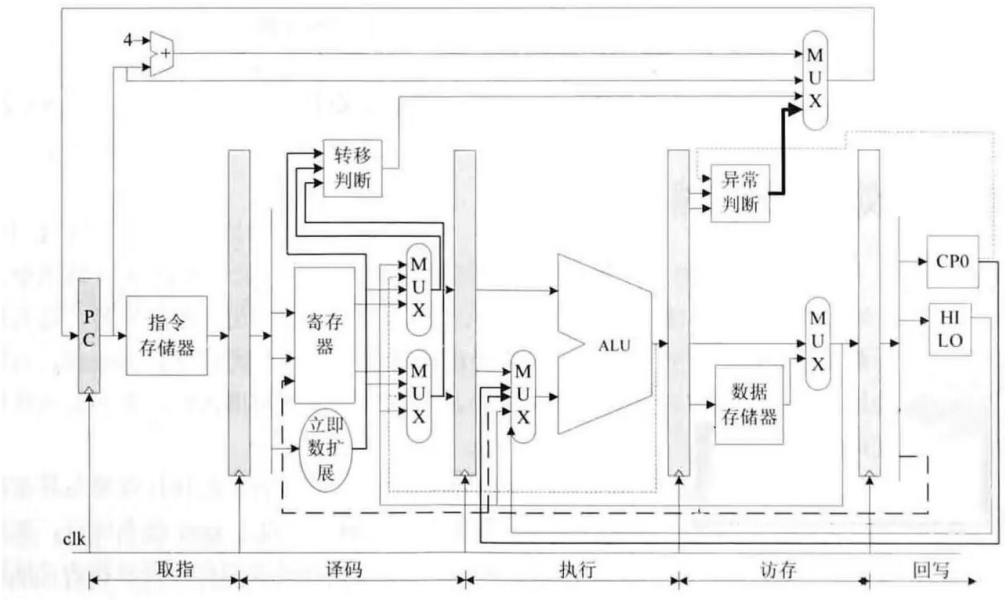


图 1: 通路结构

取指阶段分析：

取指阶段的功能是从存储器中取出下一条待执行的指令。该阶段根据指令计数器(PC)的值,从指令存储器中读取相应地址的指令,并将其传送给译码阶段。同时,PC 的值需要更新为下一条指令的地址,通常是当前地址加 4(因为 MIPS 指令的长度是 32 位,即 4 字节)。在某些情况下,PC 的值可能会发生跳转,例如遇到跳转指令、分支指令、异常等,这时需要从其他来源选择正确的地址,例如跳转目标地址、异常处理程序入口地址等。

fetch 模块接口说明：

Port name	Direction	Type	Description
clk	input	wire	时钟信号
rst	input	wire	复位信号
decode_data	input	decode_data_t	译码阶段的数据，寄存器编号和数据
inst_sram_data	input	inst_sram_data_t	指令存储器的数据，指令和外部中断信号
fetch_data	output	fetch_data_t	包含 PC 值、指令和外部中断信号
fetch_data_out	output	fetch_data_out_t	传送给 PC 模块的数据，包含 PC 值
decode_sign	input	control_sign_t	译码阶段的控制信号，各种使能和选择信号
memory_sign	input	control_sign_t	内存阶段的控制信号，包含分支标志和分支目标地址
stall	input	stall_sign_t	控制器的暂停信号，包含各个阶段的暂停信号
pc_exception	input	word_t	控制器的异常信号，表示异常处理程序的入口地址

译码阶段分析：

译码阶段主要是对取指阶段传来的指令进行解码，根据指令的类型和格式，确定指令所需的操作数和操作码，各类指令的控制信号由 main_decode 模块产生。通过访问通用寄存器，从中读取或写入指令所需的数据。同时还需要判断是否发生分支跳转，如果是，需要计算分支目标地址，并传送给取指阶段。

decode 模块接口说明：

Port name	Direction	Type	Description
clk	input	wire	时钟信号
rst	input	wire	复位信号
decode_sign	input	control_sign_t	译码阶段的控制信号

Port name	Direction	Type	Description
execute_sign	input	control_sign_t	执行阶段的控制信号
memory_sign	input	control_sign_t	内存阶段的控制信号
wb_sign	input	control_sign_t	写回阶段的控制信号
control_data	input	control_data_t	控制器的控制数据
decode_data_in	input	fetch_data_t	取指阶段的数据，包含 PC 值、指令和外部中断信号
execute_data	input	execute_data_t	执行阶段的数据，包含寄存器编号和数据
memory_data	input	memory_data_t	内存阶段的数据，包含寄存器编号和数据
wb_data	input	wb_data_t	写回阶段的数据，包含寄存器编号和数据
decode_data	output	decode_data_t	传送给执行阶段的数据，包含寄存器编号、数据、指令、PC 值和外部中断信号
stall_reqD_load	output	logic	传送给控制器的暂停请求信号，表示是否需要暂停译码阶段

译码阶段的主要模块有：

regfile：一个 32 个 32 位的通用寄存器，用于存放指令所需的数据。该模块需要从 main_decode 模块中提取立即数 imm 和同时此模块还处理了部分数据冒险，比如添加了 alu-alu 旁路和 mem-alu 旁路，以及对于 lw 类型的的数据冒险，在 E 阶段使用 stall 来暂停流水线阻塞一个周期等待 M 阶段取出数据。

对 decode_data 进行赋值，用于对取指阶段传来的指令进行解析，提取出指令中的寄存器编号、指令和配置硬件中断等信息，各类指令相关的控制信号和 alu 控制信号由 main_decode 模块解码产生，并将其传送给执行阶段。

执行阶段分析：

执行阶段根据译码阶段传来的指令和数据，进行相应的算术逻辑运算、移位运算、乘除运算、跳转运算等，并将运算结果和控制信号传送给内存阶段。该阶段还处理了数据移动指令（hilo 寄存器的数据转发）的数据冒险和数据转发，以及异常和中断的检测和处理。

Port name	Direction	Type	Description
clk	input	logic	时钟信号，用于触发运算器的更新
rst	input	logic	复位信号，用于将运算器清零
execute_sign	input	control_sign_t	执行阶段的控制信号
memory_sign	input	control_sign_t	内存阶段的控制信号
wb_sign	input	control_sign_t	写回阶段的控制信号
execute_data_in	input	decode_data_t	译码阶段的数据
memory_data	input	memory_data_t	内存阶段的数据
wb_data	input	wb_data_t	写回阶段的数据
hilo_data	input	hilo_data_t	HILO 寄存器的数据
cp0_rdata	input	word_t	CP0 寄存器的数据
exception_sign	input	exception_sign_t	异常检测模块的信号

Port name	Direction	Type	Description
overflow	output	logic	溢出标志
execute_data	output	execute_data_t	传给内存阶段的数据
stall_reqE	output	logic	传给控制器的暂停请求信号
stall_reqE_div	output	logic	否需要暂停执行阶段的除法运算
exe_sram	output	memory_data_out_t	传给指令存储器的数据

执行阶段中处理了跳转指令和数据移位指令的数据转发,以及设置除法暂停信号。在处理跳转指令时,根据指令类型进行相应的数据转发操作。在移动指令数据转发阶段,根据不同的条件判断,将需要的数据从寄存器或内存中获取并转发给目标寄存器。在除法暂停信号阶段,当遇到除法指令且除法单元未准备好时,会触发暂停信号。

执行阶段的主要模块为算术逻辑运算单元 alu 和除法器 div 模块,alu 主要用于实现各种算术逻辑运算、移位运算、乘除运算、跳转运算等。div 除法器模块则用于实现 DIV 和 DIVU 指令。

访存阶段分析:

访存阶段根据执行阶段传来的指令和数据,进行相应的存储内存访问操作,包括 LOAD 加载指令、STORE 存储指令等,并将访存结果和控制信号传送给写回阶段。同时在访存阶段还需要处理异常和中断的检测和处理。

memory 模块接口说明:

Port name	Direction	Type	Description
rst	input	logic	复位信号
memory_sign	input	control_sign_t	各种使能和选择信号,以及指令类型和名称
memory_data_in	input	execute_data_t	寄存器编号、数据、指令、PC 值和外部中断信号
data_sram_data	input	data_sram_data_t	读取数据和写入数据
wb_sign	input	control_sign_t	regwrite 字段,表示是否写入寄存器

Port name	Direction	Type	Description
cp0_cause	input	cp0_cause_t	异常的类型和代码
cp0_status	input	cp0_status_t	异常的处理方式和优先级
timer_interrupt	input	logic	是否发生定时器中断
memory_data	output	memory_data_t	寄存器编号、数据、指令、PC 值和外部中断信号
exception_data	output	exception_data_t	异常的地址和数据
exception_sign	output	exception_sign_t	异常的类型和代码

访存阶段的主要模块

(1)readdata_format：根据所读存储器地址和读出存储数据，判断不同的加载指令，生成读使能，对存储数据不同的位宽进行截取再符号扩展至 32 位生成读出输出数据

(2)writedat_format：根据写入存储地址和写入存储数据，判断不同的存储指令生成写使能。对 rt 寄存器中的值从最低位开始截取不同的位宽再填至写使能对应字节中生成输出存储器写入数据。

(3)execept_check：异常检测模块，用于检测和处理访存阶段可能发生的异常和中断。该模块的功能是根据指令的类型和数据，判断是否发生地址错、溢出、对齐、系统调用、定时器等异常或中断，并输出相应的异常类型和代码，以及异常发生时的地址和数据。

写回阶段分析:

写回阶段的功能是根据访存阶段传来的指令和数据，进行相应的寄存器写入操作，包括通用寄存器、HILO 寄存器和 CP0 寄存器，并将写入结果和控制信号传送给下一条指令。

write_back 模块接口说明:

Port name	Direction	Type	Description
clk	input	logic	时钟信号
rst	input	logic	复位信号
wb_data_in	input	memory_data_t	寄存器编号、数据、指令、PC 值和外部中断信号

Port name	Direction	Type	Description
wb_sign	input	control_sign_t	regwrite 字段,表示是否写入寄存器
wb_data	output	wb_data_t	传给 HILO 寄存器的数据
wb_data_out	output	wb_data_out_t	传给通用寄存器的数据
hilo_data	output	hilo_data_t	来自 HILO 寄存器的数据

写回阶段的主要模块

hilo_reg:HILO 寄存器模块,用于实现对 HILO 寄存器的读写操作。根据访存阶段的控制信号,判断是否需要写入 HILO 寄存器,如果是,将写回阶段的数据的 hi 和 lo 字段写入 HILO 寄存器。

cp0 模块:用于实现 MIPS 处理器的协处理器 0,也称为系统控制协处理器。它主要负责处理异常、中断、特权级别、内存管理等功能,包含了一些特殊的寄存器,如 cause、status、epc、badvaddr、count、compare 等,用于存储异常原因、处理器状态、异常返回地址、错误地址、计数器、比较器等信息。它还提供了一些信号,如 cp0_rdata、cp0_cause、cp0_status、cp0_epc、timer_interrupt 等,用于与其他模块交互。

2.2.4 cache 设计

指令 cache 设计

模块申明:

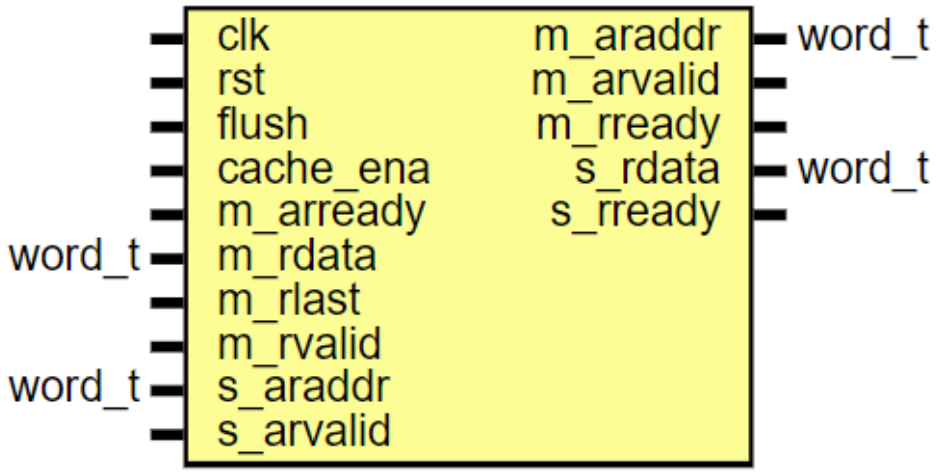


图 2

模块端口申明:

Port name	Direction	Type	Description
clk	input		时钟信号
rst	input		复位信号
flush	input		清空信号
cache_ena	input		cache 使能信号
m_araddr	output	[31 : 0]	发给内存的读地址信号
m_arvalid	output		发给地址的读地址有效信号
m_arready	input		发给内存的读地址准备信号
m_rdata	input	[31 : 0]	从内存发送过来的读数据
m_rlast	input		从内存发送过来的最后一次读信号
m_rvalid	input		从内存发送过来的读有效信号
m_rready	output		发送给内存的读准备信号
s_araddr	input	[31 : 0]	从 mips 核发送来的读地址
s_arvalid	input		从 mips 核发送来的读有效信号
s_rdata	output	[31 : 0]	发送给 mips 核的读数据
s_rready	output		发送给 mips 核的读准备信号

本模块是 MIPS 处理器中的存储缓存模块,它负责将 MIPS 处理器访问的地址分解成索引和偏移量,然后根据地址是否命中缓存来决定是直接从缓存中读取数据还是向内存发送请求。总之,这个模块的主要作用是实现缓存机制,提高数据访问速度。

数据 cache 设计

模块申明:

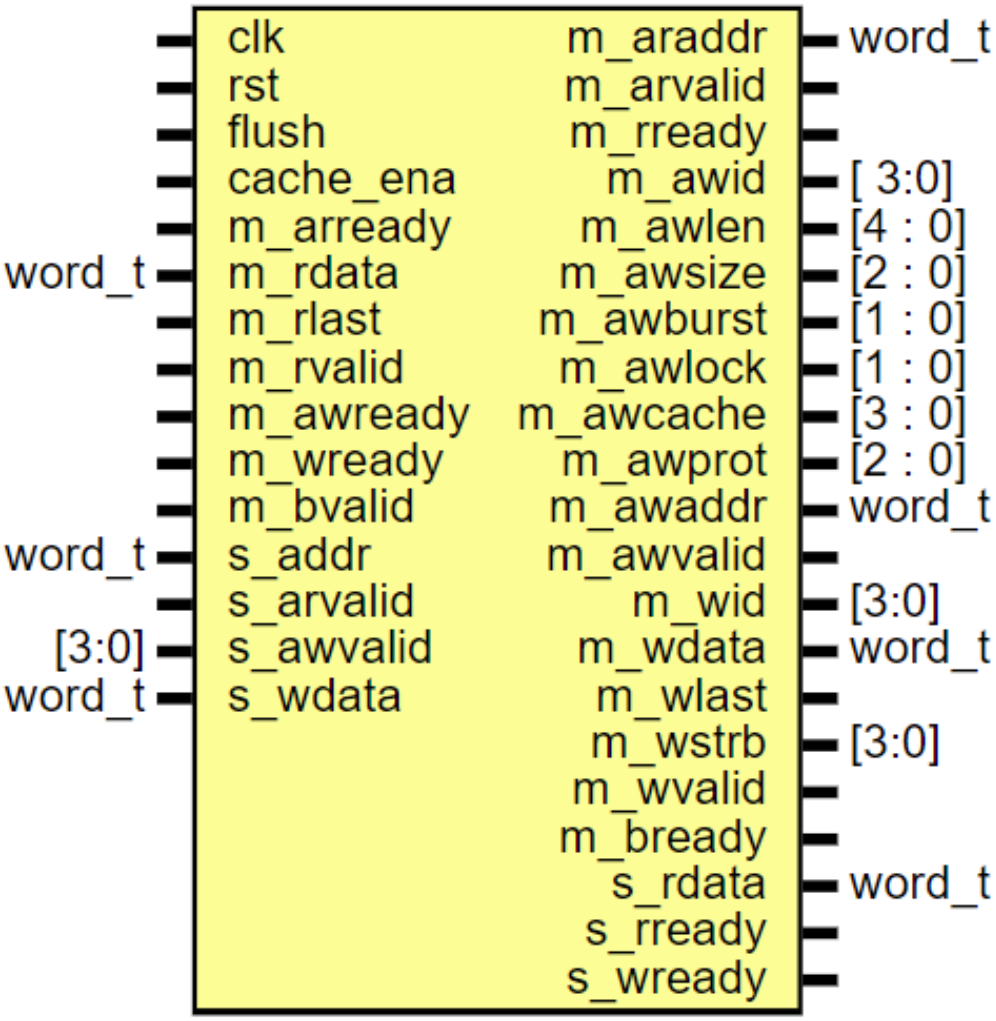


图 3

模块端口申明:

Port name	Direction	Type	Description
clk	input		时钟信号
rst	input		复位信号
flush	input		清空信号
cache_ena	input		cache 使能信号
m_araddr	output	[31 : 0]	发给内存的读地址信号
m_arvalid	output		发给地址的读地址有效信号

Port name	Direction	Type	Description
m_arready	input		发给内存的读地址准备信号
m_rdata	input	[31 : 0]	从内存发送过来的读数据
m_rlast	input		从内存发送过来的最后一次读信号
m_rvalid	input		从内存发送过来的读有效信号
m_rready	output		发送给内存的读准备信号
m_awid	output	[3 : 0]	发送给内存的写地址标志
m_awlen	output	[4 : 0]	发送给内存的写地址长度信号
m_awsize	output	[2 : 0]	发送给内存的写地址尺寸
m_awburst	output	[1 : 0]	发送给内存的写地址 burst 传输模式
m_awlock	output	[1 : 0]	发送给内存的写地址锁信号
m_awprot	output	[2 : 0]	发送给内存的写地址端口信号
m_awaddr	output	[31 : 0]	发送给内存的写地址
m_awready	input		从内存读入的写地址准备信号
s_araddr	input	[31 : 0]	从 mips 核发送来的读地址
s_arvalid	input		从 mips 核发送来的读有效信号
s_rdata	output	[31 : 0]	发送给 mips 核的读数据
s_rready	output		发送给 mips 核的读准备信号

本模块是一个用于实现写直通的数据缓存模块,支持读写操作,包括读写地址握手、读数据握手、写地址握手、写数据握手、写响应握手等功能。同时,它实现了基于 Cache 的

数据存储和访问，包括 Cache 的配置、Cache 存储单元、访问地址分解、Cache line 的读写等操作。模块中还包含了状态机和相应的状态转移逻辑，以实现在不同情况下的数据处理和 Cache 操作。

2.2.5 AXI 接口设计（mycpu-top）

模块设计如下，具体端口信号说明见模块信号申明表。

1）模块设计表：

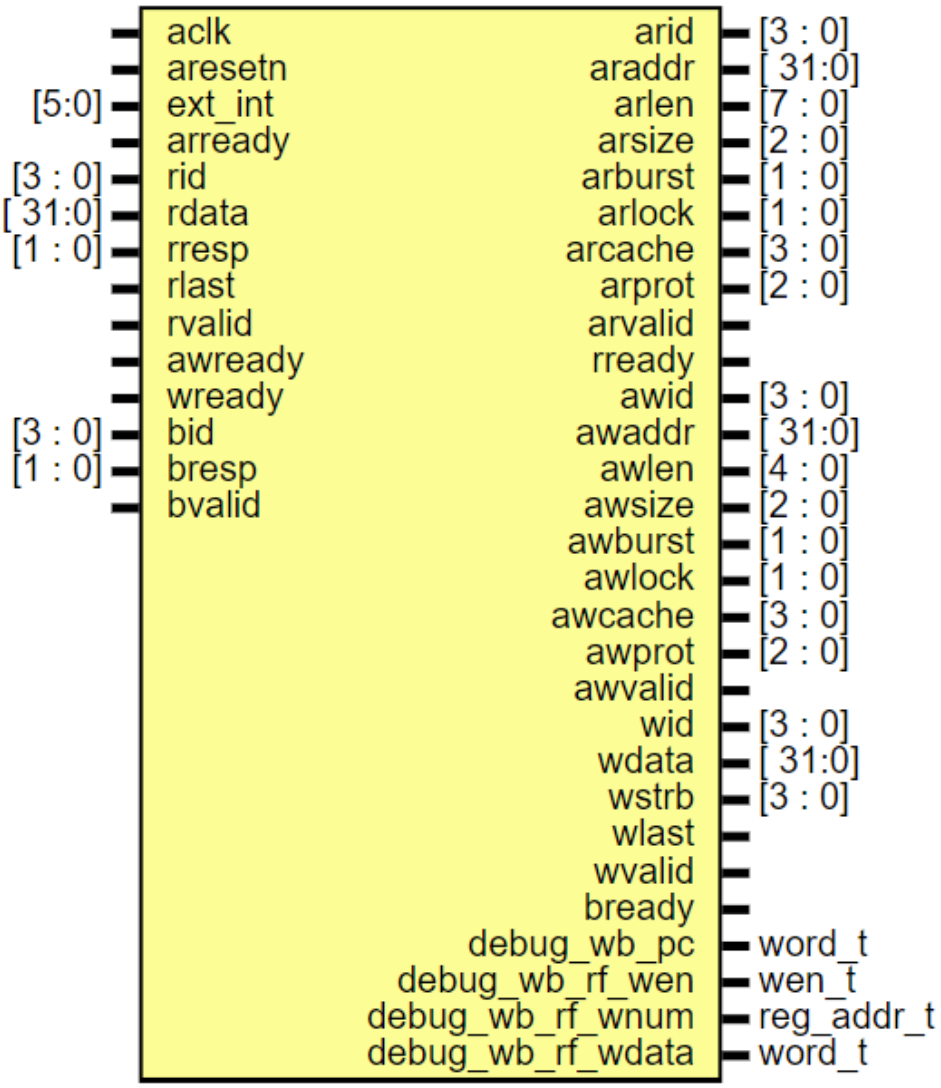


图 4

2）模块信号申明表

Port name	Direction	Type	Description
aclk	input		时钟信号
aresetn	input		复位信号
ext_int	input	[5:0]	时钟中断信号
arid	output	[3:0]	读标志信号
araddr	output	[31:0]	读地址信号
arlen	output	[7:0]	读地址长度信号
arsize	output	[2:0]	读地址尺寸信号
arburst	output	[1:0]	读地址 burst 传输信号
arlock	output	[1:0]	读地址锁信号
arcache	output	[3:0]	读地址 cache 种类信号
arprot	output	[2:0]	读地址端口信号
arvalid	output		读地址有效信号
arready	input		读地址准备信号
rid	input	[3:0]	读标志信号
rdata	input	[31:0]	读数据
rresp	input	[1:0]	读响应信号
rlast	input		读最后标志信号
rvalid	input		读有效信号
rready	output		读准备信号
awid	output	[3:0]	读有效信号

Port name	Direction	Type	Description
awaddr	output	[31:0]	写地址信号
awlen	output	[4:0]	写地址长度信号
awsize	output	[2:0]	写地址尺寸信号
awburst	output	[1:0]	写地址 burst 传输信号
awlock	output	[1:0]	写地址锁信号
awcache	output	[3:0]	写地址 cache 种类信号
awprot	output		写地址端口信号
awvalid	output		写地址有效信号
wid	output	[3:0]	写准备信号
wdata	output	[31:0]	写数据信号
wstrb	output	[3:0]	写位置信号
wlast	output		写最后传输信号
wvalid	output		写有效信号
wready	output		写准备信号
bid	input	[3:0]	响应标志信号
bresp	input	[1:0]	写响应信号
bvalid	input		写响应有效信号
bready	output		写响应有效信号

本模块是一个 MIPS 处理器的外设接口模块,它负责与外部 AXI 总线进行交互。在这个模块中,我们定义了 AXI 总线的输入和输出信号,包括地址、数据、控制等。同时,我们还定义了一些内部信号,用于表示缓存操作的状态和数据缓存和 AXI 总线的连接状态。

在这个模块中,我们使用了两个缓存模块,一个是数据缓存,另一个是指令缓存。数据缓存负责缓存来自 AXI 总线的数据,而指令缓存负责缓存来自 AXI 总线的指令。这两个缓存模块都使用相同的连接状态信号进行连接。

在连接 AXI 总线时,我们需要定义一些信号,用于表示 AXI 总线的地址、数据、控制等。这些信号包括:

① 写地址:来自 AXI 总线的写地址信号。

② 写数据:来自 AXI 总线的写数据信号。

③ 写响应:来自 AXI 总线的写响应信号。

④ 读地址:来自 AXI 总线的读地址信号。

⑤ 读数据:来自 AXI 总线的读数据信号。

⑥ 读响应:来自 AXI 总线的读响应信号。

这些信号被传递给数据缓存和指令缓存模块,然后分别处理这些信号。数据缓存模块负责将来自 AXI 总线的数据缓存到内存中,而指令缓存模块负责将来自 AXI 总线的指令缓存到指令寄存器中。

在处理 AXI 总线的请求时,我们需要定义一些信号,用于表示请求的地址、长度、大小、burst 等。这些信号包括:

① 写地址:来自 AXI 总线的写地址信号。

② 写数据:来自 AXI 总线的写数据信号。

③ 写长度:来自 AXI 总线的写长度信号。

④ 写大小:来自 AXI 总线的写大小信号。

⑤ 写 burst:来自 AXI 总线的写 burst 信号。

⑥ 写锁定:来自 AXI 总线的写锁定信号。

⑦ 写缓存:来自 AXI 总线的写缓存信号。

⑧ 写保护:来自 AXI 总线的写保护信号。

这些信号被传递给数据缓存模块,然后处理这些信号。数据缓存模块根据请求的地址、长度、大小、burst 等信息,将请求转发给内存系统或者缓存。

最后,这个模块还需要一些其他信号,用于表示缓存操作的状态、AXI 总线的连接状态、缓存操作的请求等。这些信号包括:

缓存连接状态:表示缓存与 AXI 总线是否连接的信号。

缓存操作状态:表示缓存操作进行到哪个阶段的信号。

缓存操作请求:来自 AXI 总线的缓存操作请求信号。

这些信号被传递给数据缓存和指令缓存模块,然后根据不同的信号进行处理。最后,处理后的数据和指令被传递给 AXI 总线,从而完成与外部内存系统的交互。

3 设计过程

3.1 设计流水账

(1) 2023/12/26 9:30am-11:50am,14:25pm-17:30pm: 下载软件,配置环境。查看实验操作文档,复习计算机组成原理实验过程。观看讲解视频,大致了解了要实现 52 条指令的流程。

(2) 2023/12/27 10:30am-11:50am,14:25pm-17:30pm: 复习计算机组成原理四个实验,对着数据通路图看 lab4 代码,设计需要加的具体指令,并初步进行分工。

(3) 2023/12/28 9:30am-11:50am,14:25pm-17:30pm: 阅读《自动动手写 CPU》,学习添加指令流程,观看 b 站教学视频学习 System Verilog 语言。同时根据《自己动手写 CPU》修改 lab4 代码结构,初步定义各阶段信号等结构体信息。

(4) 2023/12/29 10:00am-12:30pm,14:25pm-17:30pm: 阅读老师和助教们提供的材料,进一步理解了如何设计自己的 cpu。共同开始添加简单指令,邹奕主要添加逻辑指令,张婷婷主要添加除乘除法之外的简单算术指令,主要修改部分包括译码和 alu 对应运算。

(5) 2024/1/1 10.00am-12.30pm,14:25pm-16:00pm: 逻辑指令通过测试后邹奕添加数据移动指令并通过测试。张婷婷添加乘法指令与访存指令,陈涛添加除法指令,苟豪爽协助并推进小组进度。

(6) 2024/1/2 9:30am-11:30pm,14:30pm-17:20pm,19:00pm-22:00am: 邹奕添加移位指令;张婷婷观看视频内容,debug 访存指令,陈涛观看视频内容 debug 除法指令,并添加分支跳转指令,苟豪爽协助并推进小组进度。

(7) 2024/1/3 8:30am-11:30pm,14:30pm-17:20pm,19:00pm-22:00am: 张婷婷 debug 并实现了访存指令,陈涛添加分支跳转指令,通过了 independent 相应测试。

(8) 2024/1/4 8.30am-11.30pm,14:30pm-17:20pm,19:00pm-22:00am: 张婷婷添加自陷指令,陈涛添加特权指令,邹奕和苟豪爽协助。

(9) 2024/1/5 8.30am-11.30pm,14:30pm-17:20pm,19:00pm-22:00am: 苟豪爽连接 Sram 接口,根据功能测试结果进行 debug,其他人协助。

(10) 2024/1/6 8.30am-11.30pm,14:30pm-17:40pm,19:00pm-22:00am: 调通 SOC,功能测试

仿真 89 条指令测试通过,上板测试 debug 通过。添加基础 cache 功能模块。

(11) 2024/1/7 8.30am-11.30pm,14:30pm-17:40pm,19:00pm-22:00am: 苟豪爽调整 cache 模块,其他人协助。

(12) 2024/1/8 8.30am-12.30pm,14:30pm-17:40pm,19:00pm-23:00am: 苟豪爽连接 Axi 接口并 debug,其他人协助。

(13) 2024/1/9 8.30am-12.30pm,14:30pm-17:40pm,19:00pm-23:00am: 上板最终测试,debug。

(14) 2024/1/10 8.30am-12.30pm,14:30pm-17:40pm,19:00pm-23:00am: 小组撰写报告文档。

(15) 2024/1/11 8.30am-12.30pm,14:30pm-17:40pm,19:00pm-23:00am: 撰写报告,汇总检查讨论答辩相关事宜。

3.2 错误记录

3.2.1 错误 1

(1) 错误现象:更换不同指令 coe 文件后仿真时输入指令集仍为上一类指令 coe 文件中的指令。
(2) 分析定位过程:对比仿真结果中的指令数据与上一类指令数据一致,重新仿真后仍然不变。
(3) 错误原因:没有删除 ip 核重置存储器。
(4) 修正效果:经询问同学并查阅文档资料后对 inst 的 ip 核进行 Reset 刷新并重新生成 ip 核,然后重新运行仿真解决问题。

3.2.2 错误 2

(1) 错误现象:算术运算仿真测试时即使点击了运行按钮也会在 10000ns 时停止运行后面的所有指令。
(2) 分析定位过程:对比应该有的指令数据,查看控制台打印的错误信息为“FATALERROR: Iteration limit 10000 is reaced. Possible zero delay ocillation detected where simulation time can not advance. Peseacect your source code. ”后在网上查询资料发现可能为死循环调用导致超时,不断定位修改新添加的指令后发现时钟流动成功。
(3) 错误原因:暂时还不知道具体原因。
(4) 修正效果:删除了 alu 模块中 overflow 的赋值后正常运行。

3.2.3 错误 3

(1) 错误现象:访存指令测试时只运行了前几个指令,后面指令取不出来。
(2) 分析定位过程:观察 PC 发现 PC 只变化了一次(+4)后便一直不变,询问队友后提示可能是 stall 出现了问题。观察各个阶段的 stall 信号发现访存指令的 stall 没有正确赋值导致其为 X。
(3) 错误原因:访存指令的暂停信号没有正确赋值导致流水线无法继续流通。
(4) 修正效果:修改 stall 信号的代码后流水线可以流通,正常取出后面的指令,PC 也能正常变化。

3.2.4 错误 4

(1) 错误现象:访存指令测试时无法取出指令 iram 和数据 dram 中的值,均为 X 或 Z 值。
(2) 分析定位过程:顺着指令和数据的调用信息找到顶层模块查看接口是否连接错误。
(3) 错误原因:顶层接口被修改了导致错误取指和取数据。
(4) 修正效果:修改接口后能够成功取出数据和指令。

3.2.5 错误 5

(1) 错误现象:在仿真时,发现 LUI 指令的结果不正确,无法将立即数左移 16 位并赋值给目标寄存器。
(2) 分析定位过程:看了数据通路和控制信号后,发现没有问题,然后检查了 LUI 指令的语法和格式,发现在赋值语句中,使用了右移运算符(»),正确的应该是左移运算符(«)。
(3) 错误原因:由于没有注意运算符的方向,导致 LUI 指令的功能与预期不符。
(4) 修正效果:将右移运算符改为左移运算符,重新编译和仿真,发现 LUI 指令的结果正确了。

3.2.6 错误 6

(1) 错误现象:SLLV 指令的移位位数不能超过 5 位,否则会报错。
(2) 分析定位过程:SLLV 是根据寄存器中的值来移位的。其寄存器的位宽是 32 位,而在移位模块中,使用了一个 5 位的变量来接收寄存器的值,导致高位被截断。
(3) 错误原因:由于没有考虑到寄存器的位宽和移位的范围,导致 SLLV 指令的功能受到限制。
(4) 修正效果:将移位模块中的变量改为 32 位的,结果就正确了。

3.2.7 错误 7

(1) 错误现象:MFHI 指令和 MFLO 指令的结果不正确,无法从 HI 和 LO 寄存器中读取数据。
(2) 分析定位过程:检查 HI 和 LO 寄存器的代码,发现在读取数据时,使用了一个错误的使能信号,导致数据没有输出。
(3) 错误原因:使用了一个错误的使能信号,导致 MFHI 指令和 MFLO 指令的功能与预期不符。
(4) 修正效果:将错误的使能信号改为正确的使能信号,重新编译和仿真,结果正确。

3.2.8 错误 8

(1) 错误现象:MTHI 指令的执行结果不正确,未能将寄存器中的值正确写入 HI 寄存器。

(2) 分析定位过程:详细检查 MTHI 指令的数据移动模块,发现在写入 HI 寄存器时,使用了 rd 寄存器的值作为数据源。
(3) 错误原因:未注意数据源的选择,导致 MTHI 指令的数据移动模块将 rd 寄存器的值写入 HI 寄存器。
(4) 修正效果:将数据源从 rd 改为 rs, 修正后成功将 rs 寄存器中的值写入 HI 寄存器。

3.2.9 错误 9

(1) 错误现象:除法器除数为 0
(2) 分析定位过程:除法除数为 0 错误,就想起特殊情况没有补全
(3) 错误原因:除法器实现时忽略了除数为 0 特殊情况
(4) 修正效果:在状态机中添加除数为 0 状态,补充状态转移实现修正。

3.2.10 错误 10

(1) 错误现象: 除法器执行错误
(2) 分析定位过程: 测试了一下除法器发现单独除法指令执行过程正确,除法器模块应该没问题,检查执行错误的地方,发现涉及分支跳转附近,由于分支跳转有延迟槽,它跟着也会被执行如果不管就会导致后续指令执行出错或者计算出结果后面结果错误
(3) 错误原因: 执行的时候因为 MIPS 架构延迟槽,作为延迟槽指令如果跳转结果不作数,但是这样除法器也会执行对后续指令执行的正确执行产生影响,所以首先得对中止取消信号 annuldiv 做说明

(4) 修正效果: 在 maindecodes 中一开始就对这个信号加以说明,这样延迟槽指令就会及时终止除法器

3.2.11 错误 11

(1) 错误现象:跳转指令错误
(2) 分析定位过程:检查对应跳转条件定义,就发现了这个问题
(3) 错误原因:跳转条件中判断表达错误,对应指令参考文档写的时候转换错误
(4) 修正效果:修改了对应的判断表达,并检查了一下其他跳转条件判断是否存在写错条件。

3.2.12 错误 12

(1) 错误现象:跳转地址错误
(2) 分析定位过程:仔细阅读了一下指令定义参考文档,发现里面又一个细节,详细说明和操作定义在 PC 上有点区别。
(3) 错误原因:详细说明中是“加上该分支指令对应的延迟槽指令的 PC ”但是由于写的时候直接看操作定义“PC ← PC+ targetoffset”没去看详细描述说明就直接用 PC 但是实际上是 pcplus4
(4) 修正效果:修改了对应跳转地址的定义

4 设计结果

4.1 设计交付物说明

```
目录层次
 mycpu_top.sv                    //顶层模块
|    defines.vh                  //宏定义文件
|    mips.sv                     //mips核顶层模块
|  |    main_decode.sv           //译码器模块
|  |    datapath.sv              //数据通路模块
|  |  |    fetch.sv              //取值阶段模块
|  |  |  |    pc_reg.sv          //pc模块
|  |  |     pipeline .sv         //流水线寄存器模块
|  |  |    decode.sv             //decode模块
|  |  |  |    regfile.sv         //寄存器堆模块
|  |  |    execute.sv            //执行阶段模块
|  |  |  |    alu.sv             //alu模块
|  |  |  |    div.sv             //除法器模块
|  |  |    memory.sv             //内存模块
|  |  |  |    readdata_format.sv  //读数据格式化模块
|  |  |  |    writedat_format.sv  //写数据格式化模块
```

```
|   |   |   |    exception_check.sv     //异常检测模块
|   |   |    writeback.sv             //写回模块
|   |   |   |    hilo_reg.sv           //hilo寄存器模块
|   |   |    new_cp0.sv               //cp0模块
|   |   |    ctrl.sv                  //控制模块
|   |    sram_interface.sv            //接口转换器模块
|   |    i_cache.sv                   //指令cache模块
|   |    d_cache.sv                   //数据cache模块
|   |    axi_cache_merge.sv           //指令,数据cache合并模块
 rtl
|    myCPU
|    xilinx_ip
|    soc_lite_top.v
```

4.2 设计演示结果

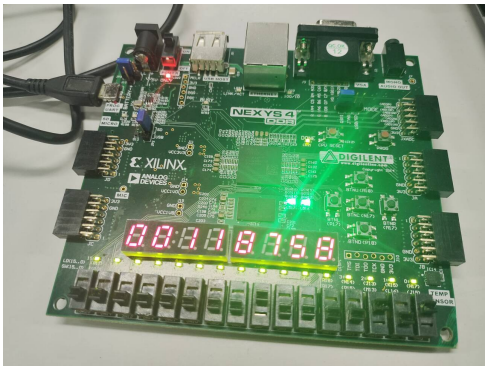
4.2.1 综合仿真测试

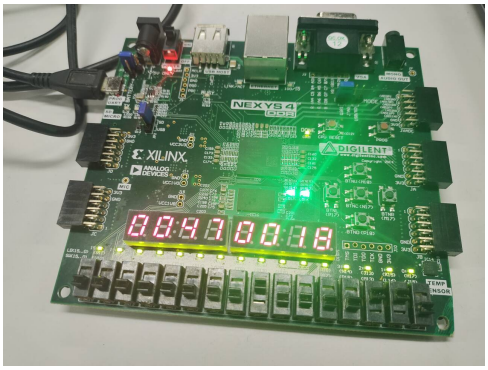
仿真测试结果如下图：

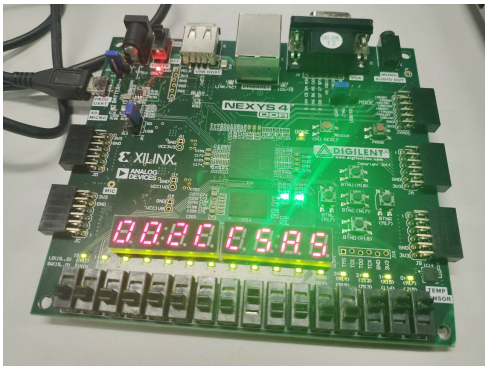
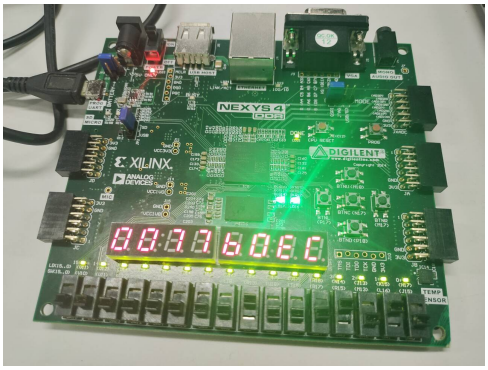
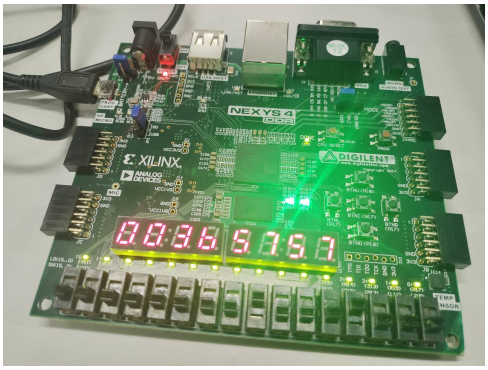
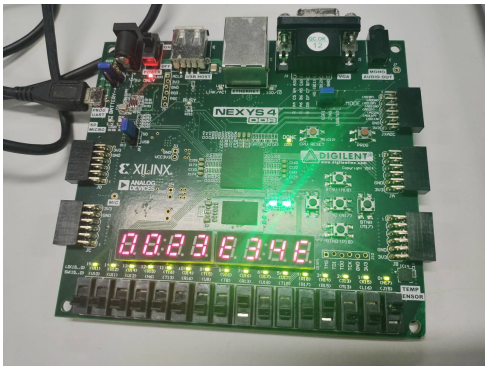
```
----[9078935 ns] Number 8'd19 Functional Test Point PASS!!!
        [9082000 ns] Test is running, debug_wb_pc = 0x00000000
        [9092000 ns] Test is running, debug_wb_pc = 0xbfc25058
        [9102000 ns] Test is running, debug_wb_pc = 0xbfc25114
        [9112000 ns] Test is running, debug_wb_pc = 0xbfc251dc
        [9122000 ns] Test is running, debug_wb_pc = 0xbfc2529c
        [9132000 ns] Test is running, debug_wb_pc = 0xbfc25364
----[9135465 ns] Number 8'd20 Functional Test Point PASS!!!
        [9142000 ns] Test is running, debug_wb_pc = 0xbfc0092c
==================================================================

Test end!
----PASS!!!
```

4.2.2 上板测试

上板测试结果如下图：





4.2.3 性能得分记录

序号	测试程序	myCPU 上板计时(16进制) 数码管显示	gs132 上板(16进制) 数码管显示	T_{gs132}/T_{mycpu}
cpu_clk : sys_clk		50MHz : 100MHz	50MHz : 100MHz	–
1	bitcount	0016CC56	13CF7FA	13.90323954
2	bubble_sort	00898EE9	7BDD47E	14.40720678
3	coremark	0094291D	10CE6772	29.0387871
4	crc32	0053C76A	AA1AA5C	32.48626419
5	dhrystone	00118758	1FC00D8	28.98125283
6	quick_sort	00470018	719615A	25.59677332
7	select_sort	002CC5A5	6E0009A	39.31029272
8	sha	003B5757	74B8B20	31.47129417
9	stream_copy	0023E34E	853B00	3.712405451
10	stringsearch	0077B0EC	50A1BCC	10.77865079

性能分	19.237

5 参考设计说明

5.1 除法状态机参考说明:

除法状态机参考了学校硬件综合设计官方实验包中给出的参考试除法的代码

5.2 cp0 寄存器参考说明:

cp0 寄存器参考了学校硬件综合设计官方实验包中给出的参考代码

5.3 指令 cache 和数据 cache

指令 cache 和数据 cache 均参考了学校硬件综合设计官方实验包中给出的 cache 参考代码

5.4 仲裁模块参考说明：

本代码实现的仲裁模块参考了 https://github.com/bit-mips/cpu 中给出的 sram-interface 模块。

5.5 数据通路图参考说明：

数据通路图参考 a-simple-MIPS-CPU(https://github.com/JF2098/A-simple-MIPS-CPU)

6 现场添加指令和答辩记录

6.1 现场添加指令

6.1.1 分工情况

小组成员一起共同完成对 ABS 指令的分解、补充定位和添加

6.1.2 完成情况

指令添加：

```
"ALU_RELU": begin
    if(a[31] == 1'b1)begin
        c = ~a;
    end else begin
        c = a;
    end
end
```

```
`OP_RELU: begin
    clt.regwrite = 1'b1;
    clt.instr_name = "RELU";
    clt.reg1_read = 1'b1;
    clt.reg2_read = 1'b0;
    clt.alufunc = "ALU_RELU";
end
```

运行测试：

```
——[9078935 ns] Number 8'd19 Functional Test Point PASS!!!
       [9082000 ns] Test is running, debug_wb_pc = 0x00000000
       [9092000 ns] Test is running, debug_wb_pc = 0xbfc25058
       [9102000 ns] Test is running, debug_wb_pc = 0xbfc25114
       [9112000 ns] Test is running, debug_wb_pc = 0xbfc251dc
       [9122000 ns] Test is running, debug_wb_pc = 0xbfc2529c
       [9132000 ns] Test is running, debug_wb_pc = 0xbfc25364
——[9135465 ns] Number 8'd20 Functional Test Point PASS!!!
       [9142000 ns] Test is running, debug_wb_pc = 0xbfc0092c
================================================================

Test end!
——PASS!!!
```

6.2 现场答辩记录

6.2.1 问题 1

助教:异常和例外返回怎么实现?
苟豪爽:我们通过指令 ERET 来实现。

6.2.2 问题 2

助教:那返回地址存储在哪里?
共同回答:协处理器 cp0 的 ert。

6.2.3 问题 3

助教:请解释一下你们 cache 代码中的 m_awvalid 这个信号?
苟豪爽:这个信号是发送给内存的写地址有效信号

6.2.4 问题 4

助教:请解释一下 cache 模块中的 m_wready 信号?
苟豪爽:这个信号是内存发送过来的写准备信号

7 总结

在这次硬件综合设计中,我们小组选择了设计并实现一个基于 MIPS 指令集的流水 CPU,它能够执行 57 条指令,能够处理各种冲突和异常。这是一次对计算机组成原理知识的综合运用和深化,也是一次对硬件设计能力的挑战和提升。

在设计过程中,我们小组分工合作,互相协调,共同解决了很多困难和问题,比如如何设计数据通路和控制信号,如何使用 System Verilog 语言描述电路,如何使用综合和仿真工具进行验证和调试,如何处理数据冒险和控制冒险,如何检测和处理异常和中断等。我们通过查阅资料,向老师和助教请教,与其他小组交流讨论,逐步完善了我们的设计方案和代码。在这个过程中,我们不仅复习和巩固了计算机组成原理的基本概念和原理,而且掌握了硬件设计的基本方法和技巧,提高了自己的分析和解决问题的能力。

在设计完成后,我们对自己的设计进行了测试和评估,使用了不同的测试用例,检查了指令的执行结果和性能指标,发现了一些错误和不足,进行了修改和优化。我们发现,设计一个流水 CPU 并不是一件容易的事情,需要考虑很多细节和特殊情况,需要不断地

改进和完善,需要有耐心和毅力。同时,我们也感受到了硬件设计的乐趣和魅力,每当我们看到自己的设计能够正确地运行,就特别有成就感。

总之,这次硬件综合设计是一次非常有意义和有价值的学习经历,它让我们对计算机组成原理有了更深入的理解和应用,也让我们对硬件设计有了更多的兴趣和热情。

参考文献

[1] A simple mips cpu. https://github.com/JF2098/A-simple-MIPS-CPU.

[2] 重庆大学硬件综合设计实验文档. https://co.ccslab.cn/.

[3] D. Patterson and J. L. Hennessy. 计算机组成原理与设计:硬件/软件接口. 机械工业出版社, 北京, 2015.

[4] 雷思磊. 自己动手写 CPU. 电子工业出版社, 2014.