计算机组成原理大实验实验报告

支持指令流水的计算机系统设计与实现

计 45 班
2014011407 王晨阳
2014011408 田文龙
2014011424 何熙巽

December 3, 2016

目录

1 实验概要 3
2 总体设计 4
2.1 流水线设计 4
2.2 数据通路设计 4
3 基本功能实现 6
3.1 概述与类型对照表 6
3.2 顶层模块 mymips 6
3.3 if/id, id/ex, ex/mem, mem/wb 寄存器组 7
3.4 暂停流水线模块 8
3.5 pc 寄存器与控制器 9
3.6 控制信号产生器 9
3.7 译码器 9
3.8 分支选择器 10
3.9 冒险检测单元 10
3.10 寄存器堆 11
3.11 算术逻辑单元 11
3.12 结构冲突选择器 12
3.13 ram1 与 ram2 适配器 12
4 扩展功能实现 14
4.1 flash 引导 14
4.2 软/硬件中断 14
4.3 ps2 键盘控制器 15
4.4 VGA 显示控制器 16
5 实验结果 18
5.1 基本功能的实验结果展示 18
5.2 ps2 键盘与中断结合实现软硬件中断 18
5.3 ps2 键盘与 VGA 显示控制器结合实现文本编辑器 19
6 总结 20
附录 1 21
附录 2 22

1 实验概要

在本次实验中，我们解析了 CPU 的基本组成，运行原理以及协同工作机制，设计并实现了基于五段流水的 CPU，CPU 的主频为 25MHz，在该主频下可以正确的运行，并且实现了 CPU 对于 ram，uart 等硬件的操作与控制，完成了 CPU 的基本功能。同时我们充分利用实验平台上的资源，对该 CPU 的功能进行了拓展，实现了 cpu 对于 flash，软硬件中断，VGA 显示以及 PS2 键盘的支持，CPU 具体支持的功能如下

1. THCO-MIPS 指令集中分配给本组的 25 条基本指令与 5 条扩展指令，这 5 条扩展指令为 SLLV、SRLV、BTNEZ、SW_RS 以及 SLT
2. 通过数据前推以及插入气泡解决数据冲突和控制冲突
3. 任意次数的中断
4. FLASH 开机引导
5. ps2 键盘的输入
6. VGA 终端的显示，与 ps2 键盘结合实现一个简单的文本编辑器

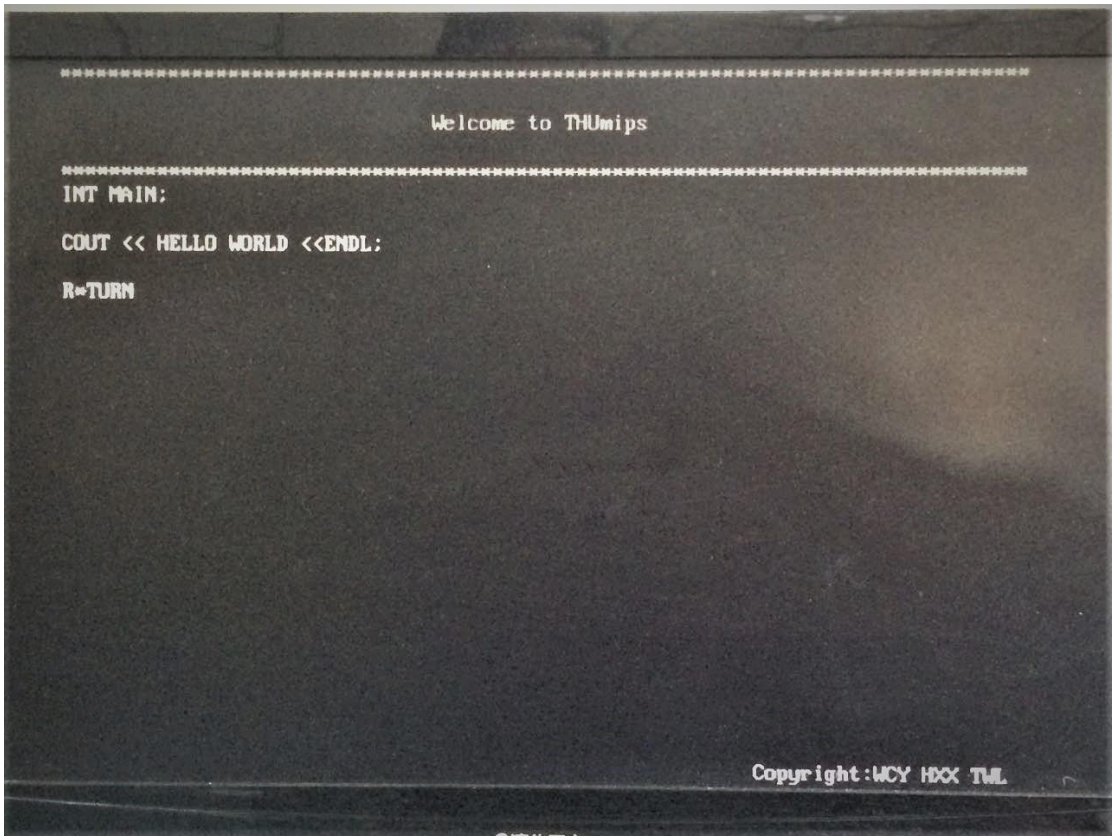

图 1：简单的文本编辑器

2 总体设计

2.1 流水线设计

2.1.1 流水线设计概述

在本次实验中，我们完成的 CPU 为支持指令流水的 CPU，采用了经典的五级流水设计，这五个阶段分别为：if（取指）、id（译码）、ex（执行）、mem（访存）以及 wb（写回）每个阶段所需要的控制信号和数据均由上一阶段产生并储存在两个阶段之间的四个段寄存器当中，这样就实现了指令控制信号的逐步流水。

2.1.2 if（取指）阶段设计

在该阶段，根据输入到 PC 寄存器控制器的控制信号在输入的数据中选择 PC 寄存器的值，并将该值以及该值对应的指令寄存器中的指令传给 if/id 寄存器组，以供下一阶段选择，输入的可供选择的数据有 PC 的当前值、PC+1 的值，分支选择器传回的值等

2.1.3 id（译码）阶段设计

在这个阶段，对 if/id 寄存器输出的指令进行译码，取出需要用到的寄存器编号以及立即数的值进行取值和立即数扩展，并且通过 controller 模块根据指令产生所有的控制信号，并将这些信息传给 id/ex 寄存器，以供下一阶段使用。同时 J 类型和 B 类型指令会在该阶段通过分支选择器进行判断并传回是否 PC 的控制信号以及供选择的 PC 寄存器的值，这样解决了控制冲突。

冒险检测单元同样位于该阶段，通过检测读寄存器的编号以及下面阶段传回的写寄存器的编号，以及读写内存的地址是否有冲突。若有，则通过数据前推解决数据冲突。

2.1.4 ex 阶段设计

根据 id 阶段产生的控制信号，以及寄存器的值、扩展后的立即数等信息，ALU 模块进行对应的逻辑运算，将结果以及后面阶段的控制信号传递到 ex/mem 寄存器

2.1.5 mem 阶段设计

在这个阶段根据传入的控制信号，ex 阶段 ALU 的运算结果，对 ram 进行读写操作

为了解决结构冲突，我们使用 ram1 作为数据寄存器，ram2 作为指令寄存器。在这一阶段我们将同时把地址信息传给两个 ram 的适配器，并且根据该地址信息选择 ram 的输出。若遇到在 mem 阶段访问 ram2 的情况，我们通过暂停流水线解决该结构冲突。

2.1.6 wb 阶段设计

在这个阶段，根据前面传入的控制信号和结果，写入寄存器的编号，将结果写入寄存器堆，到此流水线结束

2.2 数据通路设计

数据通路的总体设计如下图所示

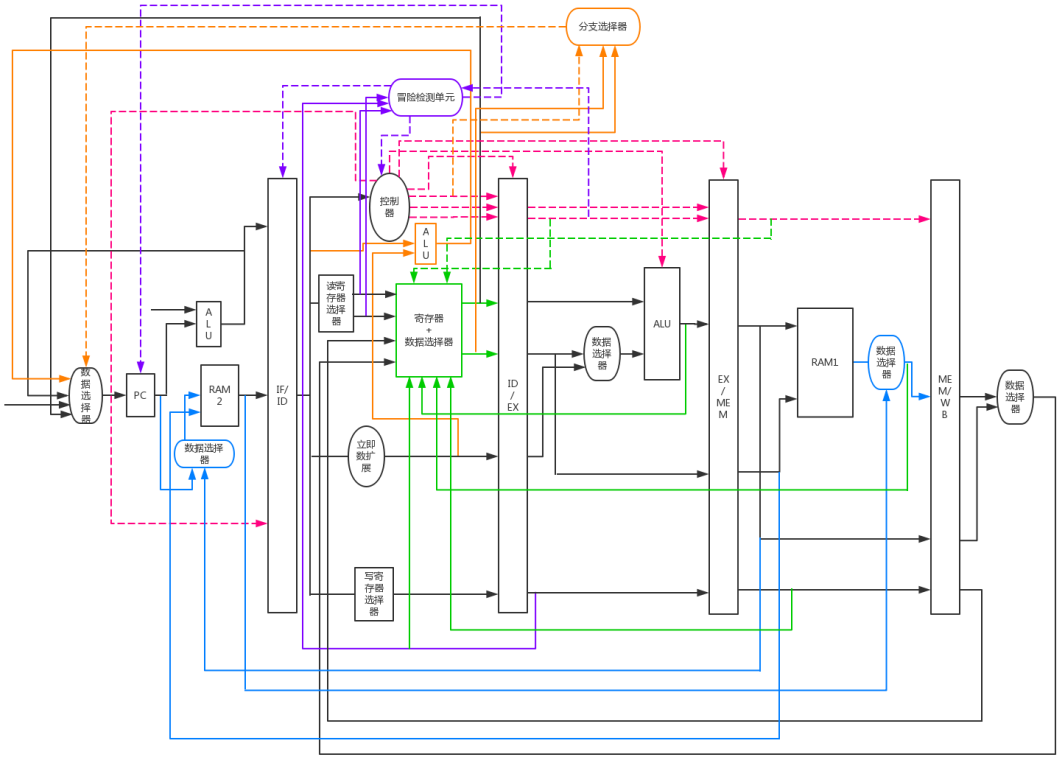

图 2：数据通路图

我们以课本上的数据通路为基础设计了该数据通路，主要的区别有

1. 取消了数据旁路单元，改为通过数据前推解决数据冲突（绿色线表示）
2. 增加了一个用以解决结构冲突的单元(蓝色线表示)，主要功能为选择两个ram1输出的数据和读取的地址
3. 将用以解决控制冲突的分支选择器独立出来作为一个单独的模块（橙色线表示）
4. 扩大了冒险检测单元的功能（紫色线表示）

3 基本功能实现

3.1 概述与类型对照表

基本功能指的是 25 条基本指令以及 5 条扩展指令的实现，指令流水结构的实现，以及控制冲突，数据冲突，结构冲突的解决。这一部分我们将主要介绍我们通过 VHDL 语言实现的该指令流水 CPU 的各个主要模块，以及这些模块间的关系。每个模块都会给出其具体功能，输入输出的信号表以及其在 CPU 结构中的位置等相关信息

为了提高代码的可读性、减少重复代码的工作量以及降低出错的几率，我们自定义了多种自定义变量，如下表所示：

类型名	内容	说明
bt	ez、nez、d、none	分支类型（零分支、非零分支，直接、非分支）
as	imme、reg、none	ALU 第二个变量来源(立即数、寄存器、无)
mtr	alu、mem、none	写回数据来源（寄存器、内存、无）
alutype	add 等	ALU 运算操作码
id_control	branch(std_logic)	是否为分支指令 0-不是 1-是
	branchtype(bt)	分支类型
	jump(std_logic)	是否为 jr 指令 0-不是 1-是
ex_control	alusrc(as)	ALU 第二个变量来源
	aluop(alutype)	ALU 运算操作码
mem_control	memread(std_logic)	是否需要读内存 0-不需要 1-需要
	memwrite(std_logic)	是否需要写内存 0-不需要 1-需要
ram_control	en(std_logic)	ram 使能信号
	oe(std_logic)	ram 读使能信号
	we(std_logic)	ram 写使能信号

表 1：自定义变量类型表

3.2 顶层模块 mymips

顶层模块的主要功能为接受时钟信号和 reset 信号以及与串口和内存交互

顶层模块的接口如下

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
tsre	in	std_logic	串口数据发送完毕标志
tbre	in	std_logic	串口发送数据标志
data_ready	in	std_logic	数据准备信号
rdn	out	std_logic	串口读使能信号
wrn	out	std_logic	串口写使能信号
ram1_control	out	ram_control	ram1 控制信号
ram2_control	out	ram_control	ram2 控制信号
ram1_data_io	inout	std_logic_vector	ram1 数据

ram2_data_io	inout	std_logic_vector	ram2 数据
ram1_addr_o	out	std_logic_vector	ram1 写地址
ram2_addr_o	out	std_logic_vector	ram2 写地址

表 2：顶层模块接口表

3.3 if/id, id/ex, ex/mem, mem/wb 寄存器组

这四个寄存器组均位于五级流水线的两个阶段之间，用以储存上一阶段的结果和向下一阶段输出控制信号和数据

If/id 寄存器将 if 和 id 两个阶段隔离开，由时钟上升沿驱动，在每个上升沿到来时，将输入信号锁存并根据上个时钟周期的输入信号改变输出信号，这里的信号主要指的是 pc 寄存器的值和指令的值，同时在这里处理关于暂停流水线的信号。

If/id 寄存器的接口如下:

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
if_pc	in	std_logic_vector	if 输入 pc 寄存器值
if_inst	in	std_logic_vector	if 输入指令值
stall	in	std_logic_vector	暂停流水线控制信号
id_pc	out	std_logic_	向 id 阶段输出 pc 寄存器值
id_inst	out	std_logic	向 id 阶段输出

表 3：if/id 寄存器接口表

id/ex 寄存器将 id 和 ex 两个阶段隔开，将 id 阶段取得或生成的数据和控制信号在时钟上升沿信号的驱动下输出到下一阶段，使得流水继续传递

id/ex 寄存器的接口如下

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
id_excontrol	in	ex_control	控制器输出信号
id_memcontrol	in	mem_control	
id_wbcontrol	in	wb_control	
stall	in	std_logic_vector	暂停流水线控制信号
id_reg1	in	std_logic_vector	rx 寄存器所存内容
id_reg2	in	std_logic_vector	ry 寄存器所存内容
id_imme	in	std_logic_vector	扩展后的 16 为立即数的值
id_regwrite	in	std_logic_vector	写回寄存器的编号
ex_excontrol	out	ex_control	向 ex 阶段输出 16 位控制信号
ex_memcontrol	out	mem_control	
ex_wbcontrol	out	wb_control	
ex_regwrite	out	std_logic_vector	向 ex 阶段输出写回寄存器编号

表 4： id/ex 寄存器接口表

ex/mem 寄存器承接 ex 与 mem 阶段，由时钟上升沿驱动，在时钟上升沿来到时将所有输入数据锁存并输出需要输出的数据，使得流水可以继续传递

ex/mem 寄存器的接口如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
ex_memcontrol	in	mem_control	上一阶段输入的控制信号
ex_wbcontrol	in	wb_control	
ex_aluans	in	std_logic_vector	alu 的运算结果
ex_aluinputb	in	std_logic_vector	alu 的第二个参数，可能在访存时使用
ex_reg_write	in	std_logic_vector	写回寄存器的编号
mem_memcontrol	out	mem_control	向下一阶段输出控制信号
mem_wbcontrol	out	wb_control	
mem_regwrite	out	std_logic_vector	向 mem 阶段输出写回寄存器编号
mem_aluans	out	std_logic_vector	向 mem 输出 alu 运算结果
mem_aluinputB	out	std_logic_vector	向 mem 输出 alu 的第二个输入

表 5：ex/mem 寄存器接口表

mem/wb 寄存器承接 wb 与 mem 阶段，由时钟上升沿驱动，在时钟上升沿来到时将所有输入数据锁存并输出需要输出的数据，使得流水可以继续传递

mem/wb 寄存器的接口如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
ex_wbcontrol	in	wb_control	上一阶段输入的控制信号
aluans	in	std_logic_vector	alu 的运算结果
ram_data	in	std_logic_vector	mem 阶段从内存中取出的结果
mem_reg_write	in	std_logic_vector	写回寄存器的编号
wb_wbcontrol	out	wb_control	向下一阶段输出控制信号
wb_regwrite	out	std_logic_vector	向 wb 阶段输出写回寄存器编号
regwritedata	out	std_logic_vector	向 wb 阶段输出写回寄存器的内容

表 6：mem/wb 寄存器接口表

3.4 暂停流水线模块

暂停流水线模块（ctrl 模块）负责处理各种各样暂停流水线的情况，将部分发来的需要暂停流水线的请求进行处理，生成 4 位暂停流水线信号码，发给执行暂停流水线的 if/id 寄存器等模块，这些模块根据 4 位编码选择暂停方式

ctrl 模块的接口如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
stallreq_form_id	in	std_logic	id 段译码后发来的暂停请求
stallreq_form_conflict	in	std_logic	因结构冲突发送的暂停流水线请求

stallreq_form_branch	in	std_logic	因 branch 类发送的暂停流水线请求
stallreq_form_jump	in	std_logic	因 jump 类发送的暂停流水线请求
stall	out	std_logic_vector	四位的暂停流水线操作码

表 7：ctrl 模块接口表

3.5 pc 寄存器与控制器

pc 寄存器与控制器（pc_reg 模块）接受所有可能的下一个 PC 值，并且通过一个控制器进行选择，用以执行跳转或是暂停流水线操作。

pc_reg 模块的接口表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
stall	in	std_logic_vector	四位的暂停流水线操作码
branchaddr	in	std_logic_vector	branch 类指令返回的地址
jumpaddr	in	std_logic_vector	jump 类指令返回的地址
pcbranchsrc	out	std_logic	是否执行 branch 类指令
pcjumpsrc	out	std_logic	是否执行 jump 类指令

表 8：pc_reg 模块接口表

3.6 控制信号产生器

控制信号产生器（controller 模块），是一个组合逻辑模块，读入指令并根据指令输出所有与其相关的控制信号，并传入到 id/ex 阶段的寄存器中

controller 模块的控制信号表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
id_inst	in	std_logic_vector	上一阶段取出的指令
control_id	out	id_control	控制器输出 id 段控制信号
control_ex	out	ex_control	控制器输出 ex 段控制信号
control_mem	out	mem_control	控制器输出 mem 段控制信号
control_wb	out	wb_control	控制器输出 wb 段控制信号

表 9：controller 模块接口表

具体的控制信号表见附录 1

3.7 译码器

译码器（interpreter 模块），是一个组合逻辑模块，读入整条指令和 PC 值，并根据指令内容输出与寄存器和立即数的内容，并连接到寄存器堆，具体来讲是两个可能的读寄存器号和一个可能的写寄存器号，以及一个扩展为 16 位的立即数，直接输出到 id/ex 寄存器

interpreter 模块的信号表如下：

信号名称	类型	信号类型	说明

rst	in	std_logic	reset 信号
id_pc	in	std_logic_vector	上一阶段取得的 pc 值
id_inst	in	std_logic_vector	上一阶段取出的指令
read_reg1	out	std_logic_vector	可能的第一个读寄存器的编号
read_reg2	out	std_logic_vector	可能的第二个读寄存器的编号
imme	out	std_logic_vector	可能的扩展后 16 位立即数的值
write_reg	out	std_logic_vector	可能的写寄存器的编号

表 10：interpreter 模块接口表

具体到每条指令的译码内容见附录 2

寄存器编号为 4 位的二进制串，若不需要该寄存器编号输出，则寄存器编号规定为“1111”，若不需要立即数输出，则立即数口输出 16 位 0

3.8 分支选择器

分支选择器（branch_ctrl 模块）是用来处理分支指令的组合逻辑模块，读入当前的 PC 值，以及经过译码器和控制信号生成模块处理过的分支指令的参数的控制信号，进行处理并判断是否进行分支，最后将结果输入到 PC 寄存器的控制器

branch_ctrl 模块的信号表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
int_pc	in	std_logic_vector	上一阶段取得的 pc 值
int_imme	in	std_logic_vector	译码器输出的立即数
int_inputA	in	std_logic_vector	寄存器堆取出的比较对象值
ctrl_branch	in	std_logic	控制信号生成模块输出，是否为分支指令
ctrl_branch_type	in	std_logic	分支指令的种类
stallreq_load	in	std_logic	若处于暂停流水线状态，暂不分支
ctrl_pc	out	std_logic_vector	执行分支指令后预期的 pc 值
branch_confirm	out	std_logic	是否进行分支

表 11： branch_ctrl 模块接口表

3.9 冒险检测单元

冒险检测单元（load_related 模块）是用来处理数据前推可能无法解决的问题，即 load 类型指令所使用的写寄存器，与下一条指令的读寄存器出现冲突这一问题，为一个组合逻辑模块，在遇到上述问题时输出暂停流水线信号到 3.4 中提到的暂停流水线模块

load_related 模块的信号表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
ex_memread	in	std_logic	上一条指令是否为 load 类型（读内存）
ex_regwrite	in	std_logic_vector	上一条指令可能写的寄存器号
id_reg1	in	std_logic_vector	id 段可能读的寄存器 1 号
id_reg2	in	std_logic	id 段可能读的寄存器 2 号
stallreq_load	out	std_logic	暂停流水线信号

表 12：load_related 模块接口表

3.10 寄存器堆

寄存器堆（regfile 模块）存放所有 13 个寄存器，编号为“0000”至“1100”，其中“0000”至“0111”对应 R0 至 R7 这 8 个通用寄存器，其余 5 个寄存器如下表

编号	“1000”	“1001”	“1010”	“1011”	“1100”
寄存器名	SP	PC	RA	T	IH

表 13：特殊寄存器编号表

寄存器堆读入译码器输出的读寄存器编号，若其合法，则输出对应寄存器的值。读寄存器的工作由组合逻辑完成同时为了解决数据冲突，数据前推的执行也在寄存器堆模块中完成。数据前推输入前两条指令所写的寄存器编号，与该阶段指令的读寄存器编号比较，若相同则跳过读寄存器阶段，通过数据前推直接输出数据，这样就替代了数据旁路模块的功能。

寄存器写回为时序逻辑模块，由时钟的上升沿驱动。值得注意的是，由于数据前推功能的存在，若 wb 阶段写寄存器与读寄存器冲突会直接进行数据前推，因此没有采用将一个时钟周期平分前一半写后一半读的设计

regfile 模块的信号表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
writeaddr	in	std_logic_vector	写端口寄存器号/数据/写使能
writedata	in	std_logic_vector	
we	in	std_logic	
readaddr1	in	std_logic_vector	读端口 1 寄存器号/使能/数据
re1	in	std_logic	
readdata1	out	std_logic_vector	
readaddr2	in	std_logic_vector	读端口 2 寄存器号/使能/数据
re2	in	std_logic	
readdata2	out	std_logic_vector	
ex_regwrite_i	in	std_logic_vector	ex 阶段数据前推
ex_writedata_i	in	std_logic_vector	
ex_regwrite_enable_i	in	std_logic	
mem_regwrite_i	in	std_logic_vector	mem 阶段数据前推
mem_writedata_i	in	std_logic_vector	
mem_regwrite_enable_i	in	std_logic	

表 14：regfile 模块接口表

3.11 算术逻辑单元

算术逻辑单元（ALU 模块）处理所有的算术逻辑运算，为组合逻辑模块，读取上一阶段输入的两个寄存器读端口的输出以及扩展后的 16 位立即数，并根据输入的控制信号，包括选择第二个操作数的 alusrc 信号和选择操作码的 aluop 信号，进行运算，最后输出结果

操作码和对应的逻辑运算内容如下表所示

(一操作数和二操作数表示直接等于第一个或第二个操作数)

操作码	addu	subu	andu	oru	xoru	sllu
逻辑运算	加	减	与	或	异或	逻辑左移
操作码	saru	srlu	sltu	equal1	equal2	equal0
逻辑运算	算术右移	逻辑右移	大小比较	一操作数	二操作数	判断相等

表 15：操作码和逻辑运算种类对照表

ALU 模块的信号如下表：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
inputA	in	std_logic_vector	可能的寄存器堆读端口 1 输出
inputB	in	std_logic_vector	可能的寄存器堆读端口 2 输出
inputC	in	std_logic_vector	可能的扩展后的 16 位立即数
excontrol	in	ex_control	控制信号
output	out	std_logic_vector	运算结果

表 16：ALU 模块接口表

3.12 结构冲突选择器

结构冲突选择器（mem 模块）负责解决在 mem 阶段读取 ram2 时出现的结构冲突，为组合逻辑模块，具体方法是将读取请求同时发给两个 ram 的适配器，然后适配器根据地址是否在该 ram 范围内输出结果至该模块，该模块在两个结果中选择正确的一个作为读取内存的输出，若产生了冲突，则输出暂停流水线信号

mem 模块的信号如下表：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
memcontrol_i	in	mem_control	控制信号
aluans_i	in	std_logic_vector	alu 的结果，即读内存的地址
ram1data_i	in	std_logic_vector	ram1 读取的结果
ram2data_i	in	ex_control	ram2 读取的结果
stallreq_ram_conflict	out	std_logic	暂停流水线信号
aluans_o	out	std_logic_vector	读内存地址输出到内存适配器
ramdata_o	out	std_logic_vector	解决结构冲突后的读内存输出

表 17：mem 模块接口表

3.13 ram1 与 ram2 适配器

ram1 适配器（ram1_adapter 模块）与 ram2 适配器（ram2_adapter 模块）为时序逻辑模块，负责处理 cpu 与 ram，cpu 与串口，ram 与串口的交互，同时 ram2 适配器也在计算机启动时直接读入 flash 中的数据，这一点将会在扩展功能的介绍中详细说明

ram1_adapter 的信号如下表

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号

clk	in	std_logic	50MHz 时钟信号
memcontrol_i	in	mem_control	四位的暂停流水线操作码
aluans_i	in	std_logic_vector	branch 类指令返回的地址
aluinputB_i	in	std_logic_vector	jump 类指令返回的地址
tsre	in	std_logic	数据发送完毕标志
tbre	in	std_logic	发送数据标志
data_ready	in	std_logic	数据准备信号
ram1_adapter_data_o	out	std_logic_vector	适配器数据输出
mem_addr_o	out	std_logic_vector	内存地址输出
mem_data_o	out	std_logic_vector	内存数据输出
ram1_data_io	inout	std_logic_vector	ram1 总线数据
rdn	out	std_logic	读使能
wrn	out	std_logic	写使能
ram2_control_o	out	ram_control	ram2 控制信号
ram1_control_o	out	ram_control	ram1 控制信号

表 18：ram1_adapter 接口表

ram2_adapter 的信号如下表：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
pc	in	std_logic_vector	pc 值
has_ram_conflict	in	std_logic	接口冲突判断
booting	in	mem_control	flash 功能用信号
flash_addr_i	in	std_logic_vector	
flash_data_i	in	std_logic_vector	
mem_addr_i	in	std_logic_vector	内存地址输出
mem_data_i	in	std_logic_vector	内存数据输出
mem_ram2_control_i	in	std_logic_vector	ram2 控制信号
ram2_adapter_data_o	out	std_logic_vector	适配器数据输出
ram2_addr_o	out	std_logic_vector	内存地址输出
ram2_data_o	out	std_logic_vector	内存数据输出
ram2_data_io	inout	std_logic_vector	ram2 总线数据
ram2_control_o	out	ram_control	ram2 控制信号

表 19：ram2_adapter 接口表

4 扩展功能实现

4.1 flash 引导

flash 引导的作用是在每次计算机启动时将监控程序直接从 flash 加载到 ram2 中，因为 flash 断电后数据不会丢失，这样可以省略每次开机时都需要重复写入监控程序的过程

具体的实现方法为将监控程序写入 flash 中，利用 flash 断电数据不丢失的特性保存监控程序的数据，在启动后启用一个计数器，随着时钟周期+1，然后将该周期读取的 flash 中的数据写入 ram2 中，直到读取完监控数据的五百余条指令后完成 flash 引导的过程，我们把这个过程叫做 booting 过程。在 booting 过程中，流水线会被暂停，过程结束后流水线启动，这时 CPU 开始运转。

flash 引导（flash_io 模块）的信号如下表：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
booting	out	std_logic	booting 过程控制信号
data_out	out	std_logic_vector	flash 输出数据
addr_out	out	std_logic_vector	flash 输出数据对应 ram2 地址
flash_byte	out	std_logic	flash 操作模式
flash_vpen	out	std_logic	flash 写保护
flash_ce	out	std_logic	flash 使能信号
flash_oe	out	std_logic	flash 读使能
flash_we	out	std_logic	flash 写使能
flash_rp	out	std_logic	flash 工作控制
flash_addr	out	std_logic_vector	flash23 位地址线
flash_data	inout	std_logic_vector	flash 输入或输出数据

表 20：flash_io 模块接口表

4.2 软/硬件中断

我们设计的中断分为指令中断和硬件中断两种类型，处理过程基本一致，但是触发方法不同，软件中断通过实现 INT 指令实现，硬件中断通过按下 PS2 键盘上的 enter 按键触发，ps2 键盘的实现将在 4.3 节中详细说明。

软硬件中断的处理过程在 if/id 寄存器中进行，收到中断信号后将该寄存器输出置 0，cpu 空转一个周期，PC 置为跳转地址，进入一状态机，该状态机具体如下表所示：

当前状态	下一状态	输出	对应指令
0001	0010	1110111001000000	MFPC r6
0010	0011	0110001111111111	ADDSP FF
0011	0100	1101011000000000	SW_SP r6 0
0100	0101	0110111001001111	LI r6 imme
0101	0110	0110001111111111	ADDSP FF
0110	0111	1101011000000000	SW_SP r6 0

0111	1000	0110111000000101	LI r6 5
1000	1001	0000100000000000	NOP
1001	1010	1110111000000000	JR r6
1010	0000	0000100000000000	NOP

表 21：中断处理状态机

该状态机即为中断处理状态机，其执行的命令如上表所示

4.3 ps2 键盘控制器

ps2 键盘控制器的设计主要由 3 个模块组成, 一个顶层模块 my_keyboard 与 CPU 交互，一个 key_data 模块，用 50MHz 驱动，接受扫描码，并将处理过的扫描码输出到 convert 模块，在这里截获 enter 键的按下和弹起事件，输出到顶层模块中再输出到 CPU，作为硬件中断的触发信号

扫描码的读取使用了栈这一数据结构的思想，在模拟 D 触发器处理 ps2_clk 的毛刺后，对于每个 ps2_clk 的上升沿，将 ps2_data 的数据压入一个大小为 12 栈中，栈低初始为 0，栈满后视为成功读取一个扫描码，这时会将同步信号置为 1，激活 convert 模块，并将处理后的扫描码中的 8 位输入 convert 模块

同时为了实现第一章中提到的文本编辑器，我们在 CPU 中添加了一个将扫描码转化为 ASCII 码的模块（ascii 模块），该模块为一个组合逻辑模块，在检测到同步信号为 1 时，将读入到的扫描码中的 8 位转化为按键对应的字母的十六进制 ASCII 码，然后通过输出给 VGA 模块，通过 VGA 控制器处理后显示在屏幕上，这一点会在 4.4 节中详细说明

my_keyboard 模块的信号表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
ps2clk	in	std_logic	ps2 键盘时钟信号
ps2data	in	std_logic	ps2 键盘数据信号
request	out	std_logic	同步信号
data	out	std_logic_vector	扫描码中的八位数据信号
con_clk	out	std_logic	enter 按键按下事件信号

表 22：my_keyboard 模块接口表

key_data 模块的信号表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
ps2clk	in	std_logic	ps2 键盘时钟信号
ps2data	in	std_logic	ps2 键盘数据信号
request	out	std_logic	同步信号
data	out	std_logic_vector	扫描码中的八位数据信号

表 23：key_data 模块接口表

convert 模块的信号表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
request	in	std_logic	同步信号
data	in	std_logic_vector	扫描码中的八位数据信号
con_clk	out	std_logic	enter 按键按下事件信号

表 24：convert 模块接口表

ascii 模块接口表

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	50MHz 时钟信号
request	in	std_logic	同步信号
data	in	std_logic_vector	扫描码中的八位数据信号
asc_clk	out	std_logic	合法按键按下事件信号
ascii	out	std_logic_vector	扫描码转换的八位 ASCII 码

表 25：ascii 模块接口表

4.4 VGA 显示控制器

VGA 显示控制器主要由四个模块组成，顶层模块（vga_top 模块）与 ps2 键盘以及 CPU 进行交互，并且通过 VGA 接口与显示器相连，在 25M 时钟的驱动下，输出 RGB 值、行同步信号、场同步信号等信息。

信号同步模块(VGA 模块)主要负责同步显示器,考虑到使用显示器分辨率为 640×480，帧频率为 60Hz，因此使用 25MHz 的时钟作为电子束扫描显示区和消隐区的扫描频率。在每行结束时对行同步信号进行同步，全部扫描完成时对场同步信号进行同步，并即时向顶层模块返回当前扫描位置的坐标

屏幕显示模块（char_mem 模块）将整个屏幕分成了 80*30 的小方块，每个方块用于存储每个位置上显示的字符的 ASCII 码值，并接受信号同步模块返回的当前扫描位置坐标，然后根据该坐标将该位置字符的 ASCII 码值传入字符模型模块。

字符模型模块（font_rom 模块）根据 16 进制 ASCII 码的顺序储存了尺寸为 8*16 的 128 个可打印字模。从屏幕系那是模块获得 ASCII 值后将该位置对应的 RGB 值返回给顶层模块，VGA 显示控制器的整个工作流程到此结束

vga_top 模块的信号表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk_out	in	std_logic	25MHz 时钟信号
char_we	in	std_logic	字符写使能
char_value	in	std_logic_vector	扫描码中的八位数据信号
r	out	std_logic_vector	rgb 值
g			
b			
hs	out	std_logic	行同步信号

vs	out	std_logic	场同步信号

表 26：vga_top 模块接口表

vga 模块的信号表如下：

信号名称	类型	信号类型	说明
rst	in	std_logic	reset 信号
clk	in	std_logic	25MHz 时钟信号
h_counter	out	std_logic_vector	行计数信号
v_counter	out	std_logic_vector	场计数信号
hs	out	std_logic	行同步信号
vs	out	std_logic	场同步信号

表 27：vga 模块接口表

char_mem 模块的信号表如下：

信号名称	类型	信号类型	说明
clk	in	std_logic	25MHz 时钟信号
char_read_address	in	std_logic	mem 读地址
char_write_address	in	std_logic_vector	mem 写地址
char_write_value	in	std_logic_vector	mem 写数据
char_read_value	out	std_logic_vector	mem 读数据
char_we	in	std_logic	mem 写使能信号

表 28：char_mem 模块接口表

font_rom 模块的信号表如下

信号名称	类型	信号类型	说明
addr	in	std_logic_vector	取字符 11 位信息(ASCII 码+4 位行数)
clk	in	std_logic	25MHz 时钟信号
char_we	in	std_logic	按行输出字符信息

表 29：font_rom 模块接口表

5 实验结果

5.1 基本功能的实验结果展示

```
>>COM5
   Ok..  Connected with com...
OK
```

图 3：成功显示 OK;
这代表监控程序中的 25 条基本指令成功实现

```
  >> A
[4000]  LI R1 1
[4001]  LI R2 2
[4002]  SLLV R2 R1
[4003]  JR R7
[4004]

  >> G
running time : 0.006 s

  >> R
R0=0000  R1=0004  R2=0002
R3=0000  R4=0000  R5=8007
```

图:4：扩展指令 SLLV 的测试样例

```
  >> A
[4000]  LI R1 1
[4001]  LI R2 2
[4002]  CMP R1 R2
[4003]  BTNEZ 1
[4004]  LI R1 2
[4005]  NOP
[4006]  JR R7
[4007]

  >> G
running time : 0.006 s

  >> R
R0=0000  R1=0001  R2=0002
R3=0000  R4=0000  R5=8007
```

图 5：扩展指令 BTNEZ 的测试样例

```
  >> A
[4000]  LI R1 1
[4001]  LI R2 2
[4002]  SLT R1 R2
[4003]  BTNEZ 1
[4004]  LI R1 2
[4005]  NOP
[4006]  JR R7
[4007]

  >> G
running time : 0.006 s

  >> R
R0=0000  R1=0001  R2=0002
R3=0000  R4=0000  R5=8007
```

图 6：扩展指令 SLT 的测试样例

```
  >> A
[4000]  LI R1 1
[4001]  LI R2 2
[4002]  SLLV R2 R1
[4003]  JR R7
[4004]

  >> G
running time : 0.006 s

  >> R
R0=0000  R1=0004  R2=0002
R3=0000  R4=0000  R5=8007
```

图 7：扩展指令 SLT 的测试样例

```
  >> A
[4000]  SW_RS 0
[4001]  LW_SP R1 0
[4002]  JR R7
[4003]

  >> G
running time : 0.006 s

  >> R
R0=0000  R1=0000  R2=0002
R3=0000  R4=0000  R5=8007
```

图 9：扩展指令 RW_RS 的测试样例

通过这 5 个测试样例，可以说明 5 条扩展指令已经全部实现并可以正常工作，这说明我们的计算机可以很好的完成所要求的基本功能

5.2 ps2 键盘与中断结合实现软硬件中断

软件中断由实现 INT 指令实现，如下右图所示，硬件中断通过截获 ps2 键盘按下并松开 enter 键触发，效果如下左图所示：

图 10：通过 enter 键触发的硬件中断

图 11：通过 int 指令触发的软件中断

处理过中断后执行相同指令所花费的时间会有毫秒级别的延长，这代表中断正确进行

5.3 ps2 键盘与 VGA 显示控制器结合实现文本编辑器

文本编辑器的主要界面如第一章图 1 所示，其中下图所示部分以及位于屏幕右下角的部分为固定在屏幕上的界面，其余空白部分可以用作文本编辑，我们一共实现了 26 个英文字母以及一些常用符号并 0~9 这 10 个数字的正确显示，并可以通过 space 键输出空格，通过 enter 键输出换行

图 12：欢迎界面及右下角的 copyright 栏

我们对 PS2 键盘处理的一点不足是没有办法处理多按键同时按下的情况，遇到这种情况时候输出的字符会显示为*，下图所示的原单词 RETURN 在输入 E 时同时按到了其它按键，这时候输出了字符*，变为了 R*TURN

图 13：在通过 PS2 键盘输入时的错误

6 总结与感想

经过了一星期的设计与两星期的实现，我们小组按时成功完成了“奋斗三星期，造台计算机”的任务。当我们最后实现的“ps2 键盘与 VGA 控制器联合文本编辑器”成功实现的时候，所有小组成员都感到如释重负。作为我们这三星期成果的这台计算机，虽然有着很多的不足，例如主频只有 25MHz，而有些小组的主频可以做到 45MHz 以上，也有着很多我们开始想实现但最终没有能够实现的功能，例如通过硬件 Term 实现用 PS2 键盘输入指令，在显示器上显示结果这一终极目标，但是总体来讲还是令人满意的，也令全部的小组成员在看到我们的实验成果时有一种成就感涌上心头。回顾这三星期的历程，毫无疑问我们是有许多收获的

首先，经过三星期，我们对于 VHDL 语言以及背后的硬件设计理论有了前所未有的了解，开始时 VHDL 作为一门语言，与我们所熟悉的 C++，JAVA，PYTHON 等高级程序设计语言的差异让我们无所适从，在完成代码之后需要几倍的时间去让它通过编译，以致最终正确运行。在大实验之前的实验 2 中，将串口实验与内存读写实验组合的失败尝试也让我们一度感到非常沮丧与无所适从。其中最主要的差异就在于其执行时的时序问题，以及芯片的硬件特性所带来的局限性。但在这三星期后，我们对于这种差异性有了一定的适应，这一点在代码的开发阶段有着充分的体现。在实现阶段的开始，我们的几乎每一个模块在测试时最初都没有办法正常运行，正常运行后也无法得到预想当中的结果，但在开发的最后阶段，每个模块开发完成后第一个版本基本上都可以完成预定的的功能，只需要对其功能上缺少的部分作出一定调整便可以投入使用，这也大大加快了我们的开发流程。

第一周的设计流程对我们来说也是宝贵的经验，在设计阶段的最初我们体会到了自己对课堂上知识掌握的程度不够，首个版本数据通路的设计，即我们在第二次小课上展示的数据通路 1.0，其过程非常的艰难，而且最终基本上与课本上的设计一致，并没有加入多少我们自己的理解，但是 3 天后的数据通路 4.0 版本，不仅根据我们自己的理解加入了大量的调整，以至于删除了一个重要的模块，而且最后证明这一设计考虑的非常充分，与最后实际实现的版本只有部分地方做出了微调。这次实验的经历让我们对课程内容有了更好的理解。

与这一学期之前的软件工程大作业一道，这次以小组为单位的实验过程，让我们对于团队协作有了更充分的理解，小组成员之间的分工泾渭分明，其工作量和工作方向也考虑到了个人的水平以及擅长的方向，同时我们也利用了 git 工具进行版本控制，解决了 VHDL 在合并时相较于其它的高级程序语言更难进行的问题。包括最终的版本在内一共有 5 次比较大的版本变化，也有了利用版本回退功能避开通过调试无法解决的问题的经历。

在这三星期的过程中我们将大部分的时间都投入到了这一实验当中，虽然已经大三了，但是再度让我们找回了刚进入大学时的冲劲与对事物的新鲜感，对之后的学习生活肯定会有很大的帮助。在此我们要感谢刘卫东老师在课堂上对理论知识的讲解，和在实验小课上对我们的数据通路和模块设计的指导，感谢周围的在我们遇到困难时帮助我们的同学，以及通过努力最终完成了这一实验的小组成员们，正是因为有这么多人的帮助与努力，才为我们带来了这一段令人难忘的经历，才让我们能够成功应对这一次令人难忘的挑战。

何熙巽 田文龙 王晨阳

2016/12/4

附录 1 控制信号表

信号 指令	ID级			EX级		MEM级		WB级	
	Branch	BranchType	Jump	ALUSrc	ALUOp	MemRead	MemWrite	RegWrite	MemtoReg
ADDIU		X	0	Imme	ADDU	0	0	1	ALU
ADDIU3	0	X	0	Imme	ADDU	0	0	1	ALU
ADDSP	0	X	0	Imme	ADDU	0	0	1	ALU
ADDU	0	X	0	Reg	ADDU	0	0	1	ALU
AND	0	X	0	Reg	ANDU	0	0	1	ALU
B	1	D	0	X	X	X	X	X	X
BEQZ	1	EZ	0	X	X	X	X	X	X
BNEZ	1	NEZ	0	X	X	X	X	X	X
BTEQZ	1	EZ	0	X	X	X	X	X	X
BTNEZ	1	NEZ	0	X	X	X	X	X	X
CMP	0	x	0	Reg	EQUAL0	0	0	1	ALU
JR	0	x	1	x	x	x	x	x	x
LI	0	x	0	Imme	EQUAL2	0	0	1	ALU
LW	0	x	0	Imme	ADDU	1	0	1	Mem
LW_SP	0	x	0	Imme	ADDU	1	0	1	Mem
MFIH	0	x	0	x	EQUAL1	0	0	1	ALU
MFPC	0	x	0	x	EQUAL1	0	0	1	ALU
MTIH	0	x	0	x	EQUAL1	0	0	1	ALU
MTSP	0	x	0	x	EQUAL1	0	0	1	ALU
NOP	0	0	0	x	X	0	0	0	x
OR	0	x	0	Reg	ORU	0	0	1	ALU
SLL	0	x	0	Imme	SLLU	0	0	1	ALU
SLLV	0	x	0	Reg	SLLU	0	0	1	ALU
SLT	0	x	0	Reg	SLTU	0	0	1	ALU
SRA	0-	x	0	Imme	SRAU	0	0	1	ALU
SRLV	0	x	0	Reg	SRLU	0	0	1	ALU
SUBU	0	x	0	Reg	SUBU	0	0	1	ALU
SW	0	x	0	Imme	ADDU	0	1	0	x
SW_RS	0	x	0	Imme	ADDU	0	1	0	x
SW_SP	0	x	0	Imme	ADDU	0	1	0	ALU

表 30：控制信号表

注：表中显示为 x 的部分在具体实现中均规定为 none

由于 INT 指令的实现不遵从五级流水过程，因而其控制信号不在表中

附录 2 译码表

指令名称	read1	read2	imme	write
ADDIU	rx		im_8	rx
ADDIU3	rx		im_4	ry
ADDSP	sp		im_8	sp
BTEQZ	t		im_8	
BTNEZ	t		im_8	
MTSP	rx			sp
SW_RS	sp	ra	im_8	
ADDU	rx	ry		rz
SUBU	rx	ry		rz
AND	rx	ry		rx
CMP	rx	ry		t
JR	rx			
MFPC	pc			rx
OR	rx	ry		rx
SLLV	rx	ry		ry
SLT	rx	ry		t
SRLV	rx	ry		ry
B			im_11	
BEQZ	rx		im_8	
BNEZ	rx		im_8	
LI			im_8	rx
LW	rx		im_8	ry
LW_SP	sp		im_8	rx
MFIH	ih			rx
mtih	rx			ih
NOP				
SLL	ry		im_3	rx
SRA	ry		im_8	rx
SW	rx	ry	im_8	
SW_SP	sp	rx	im_8	

表 31：指令译码表

注：表中 imme 列"im_"后的数字代表该条指令指令中的立即数位数
表中空白部分在实际实现中若处于 imme 列，则输出 16 位 0，否则输出 4 位 1
由于 INT 指令实现不遵照五级流水过程，因而其译码结果不在表中