五级流水线 RISC-V 处理器 实验报告

2023 秋季 计算机组成原理 大实验

组号 24

归诺祺 计 12 2021012922

方何睿 计 13 2021010756

李文赢 计 05 2020080108

第1章　实验概述

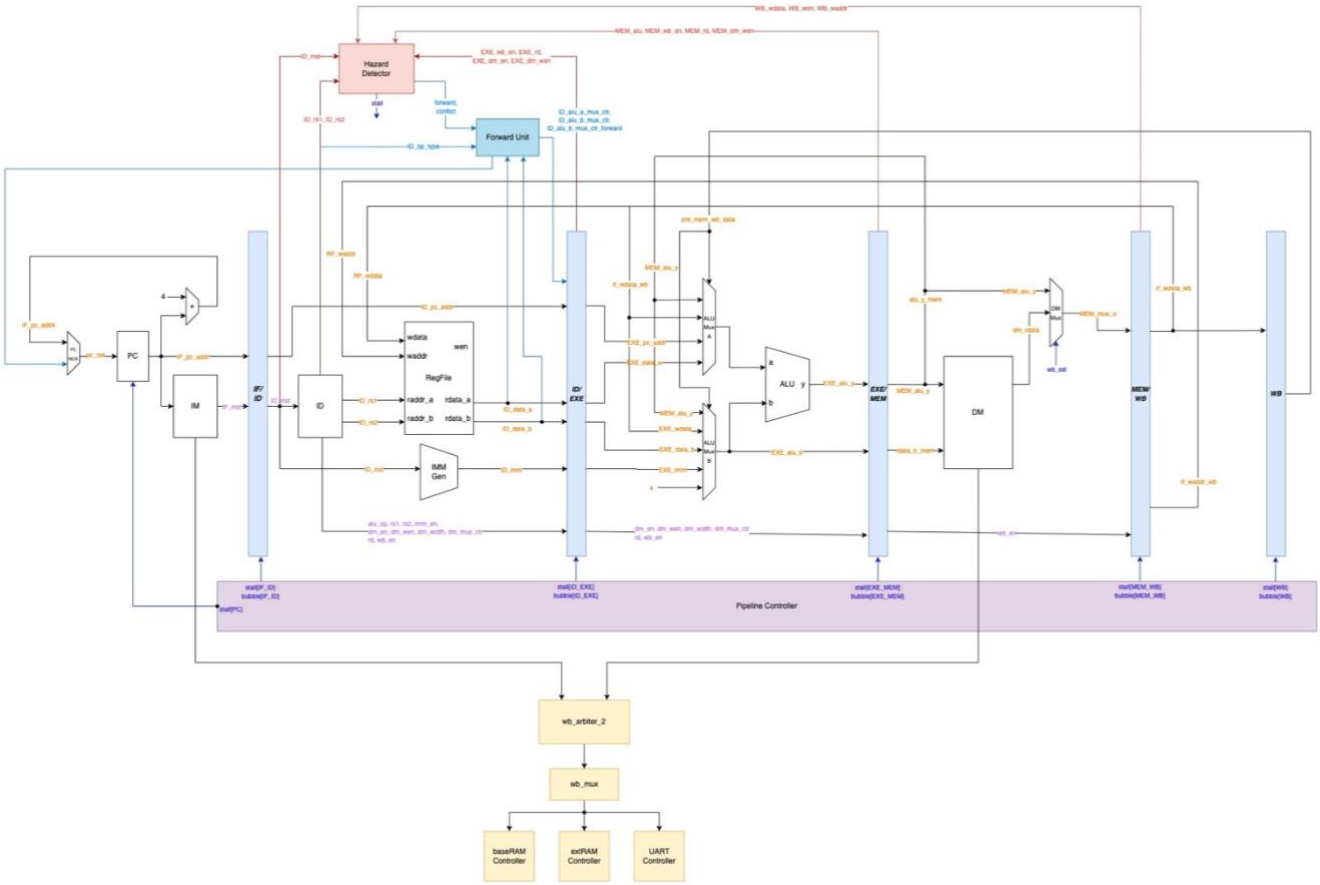
1.1　实验目的

- 深入理解流水线结构计算机指令的执行方式和基本设计方法。
- 深入理解计算机的各部件组成及内部工作原理。
- 加深对于 RV32I 指令集的理解。
- 掌握计算机外部输入输出的设计。
- 提高硬件设计和调试的能力。

1.2　实验内容

- 在实验平台上实现一个能够支持运行 RV32I 指令集的监控程序
- 实现额外三条指令：MIN、XNOR、CLZ
- 具有内存（SRAM）访问功能，实现数据与代码的存储
- 利用串口实现计算机的输入输出模块，支持与 PC 的相互通信

第2章　实验处理器设计



图表 1：数据通路

2.1 基本架构

2.1.1 阶段寄存器

- REG_IF_ID: 用于 IF 阶段和 ID 阶段之间传递数据
- REG_ID_EXE: 用于 ID 阶段和 EXE 阶段之间传递数据
- REG_EXE_MEM: 用于 EXE 阶段和 MEM 阶段之间传递数据
- REG_MEM_WB: 用于 MEM 阶段和 WB 阶段之间传递数据
- REG_WB: 存储写回阶段生成的数据，并向前传递给前面阶段

2.1.2 IF 阶段

- pc 模块：管理程序计数器或指令指针，将指令地址更新为下一个指令
- pc_mux 模块：下一条指令地址的多路选择器
- if_im 模块：通过 Wishbone 总线向存储器进行通信发出请求和取出指令

2.1.3 ID 阶段

- inst_decoder 模块：将输入指令进行解码并生成对应的控制信号
- regfile 模块：寄存器堆
- imm_gen 模块：立即数生成器

2.1.4 EXE 阶段

- alu_mux_a 模块：操作数选择器，根据控制信号确定进行计算的操作数‘a’
- alu_mux_b 模块：操作数选择器，根据控制信号确定进行计算的操作数‘b’
- alu 模块：执行算术和逻辑运算，根据操作码执行操作

2.1.5 MEM 阶段

- mem_dm 模块：通过 Wishbone 总线向存续器进行数据读写操作 Load/Store
- mem_dm_mux 模块：多路选择器，根据控制信号确定数据为输出

2.1.6 总线

- Arbiter 仲裁器：用于调节多个总线主设备对单个总线从设备的访问。管理两个 Wishbone Master（if_im、mem_dm）与 Wishbone Slave（baseRAM_controller、extRAM_controller、UART_controller）连接起来
- baseRAM_controller: 管理 BaseRAM，接受 Wishbone 总线的信号并转换成 SRAM 借口所需要的信号。内存空间为 0x8000_0000～0x803F_FFFF
- ExtRAM_controller: 管理 ExtRAM，接受 Wishbone 总线的信号并转换成 ExtRAM 借口所需要的信号。内存空间为 0x8040_0000～0x807F_FFFF
- UART_controller: 管理 UART 设备，连接到 Wishbone 总线的信号，处理 UART 借口的通信。内存空间为 0x10000000～0x1000FFFF

2.1.7 数据旁路

- forward_unit 模块：数据前递和控制流转发，以解决数据冲突和控制冲突
- 根据不同指令类型，配置控制流选择器，确定下一个指令地址
- 根据控制冲突情况，设置数据前递的选择器，以确保在执行阶段能正确获取数据
- 具体实现
  - 因为跳转指令的跳转要在 ID 阶段末就处理，所以得提前获取各个阶段末的目的寄存器的最新的值，做当前跳转指令源寄存器的值更新
  - 处理跳转地址的多路选择器的选择信号
  - 根据冲突检测传来的不同阶段的冲突信号，改变下一阶段中这些被用到的源寄存器的值的多路选择器的选择信号，来实现数据前传

2.1.8 冲突检测

- hazard_detector 模块：检测流水线中的冲突
- 根据当前阶段指令和数据操作对寄存器堆的读写情况进行判断是否存在数据冲突或需要插入气泡
- 如果是分支指令或跳转指令并且涉及到某个阶段正在写入的寄存器，判断是否需要停止，避免数据冲突

- 如果是其他指令，并且涉及到某个阶段正在写入的寄存器，标记相应阶段的数据冲突标志，用来进行后续处理
- ‘forward’开头的信号是用于在冲突或暂停发生时向前传递数据，确保能够继续流水线的执行
- 具体实现：
  - **控制冲突**
    - 本组在 ID 阶段提前处理跳转，此时 ID 阶段结束后下一时钟周期直接跳转
    - 然而跳转地址依赖的寄存器的值可能此时在 EXE 阶段准备被计算，若是如此情况，stall 一周期；同理，若依赖的寄存器的值是从 SRAM 中读取(load 指令)，或即将被写回的对应寄存器，则也 stall 一周期以取得正确的最新的数值
    - 若当前出现控制冲突，则将冲突信号 conflict_bc 传给 forward unit(管理数据旁路)
  - **数据冲突**

    判断 ID 阶段源寄存器是否与后续阶段要写回的目的寄存器相同，并将对应阶段产生的冲突信号传给数据旁路的处理程序

2.1.9 控制器

- pipeline_controller 模块：用于管理流水线的各个阶段的暂停和气泡情况
- 使用了 6 位宽度向量表示不同阶段的暂停和气泡情况
- 具体处理
  - 对于 Data memory 或 Instruction memory 未得到响应前，stall 流水线
  - dm 读取得到响应后，因为 im 仍未取出 pc 的指令(arbiter 只让其中一个 memory 工作，因此 stall ID 阶段及以前一周期等待指令读入)
  - 对冲突控制传来的需要 stall 处理的请求进行响应
  - 对于跳转指令，若 IF 周期获取的指令因为冲突失效，则插入气泡

2.2 基础版本

2.1.1 基本指令

类型	指令	branching	pc_sel	rs1	rs2	alu_a_sel	alu_b_sel	alu_op	dm_en	dm_we	regfile_we	wb_sel
R	add	-	pc+4	rs1	rs2	rs1	rs2	ADD	0	0	1	alu_y
	and							AND				
	or							OR				
	xor							XOR				
I	addi	-	pc+4	rs1	0	rs1	imm	ADD	0	0	1	alu_y
	andi							AND				
	ori							OR				
	slli							SLL				
	srli							SRL				
L	lb	-	pc+4	rs1	0	rs1	imm	ADD	1	0	1	alu_y
	lw											
S	sb		pc+4	rs1	rs2	rs1	imm	ADD	1	1	0	alu_y
	sw											
B	beq	1	alu_y	rs1	rs2	pc	imm	ADD	0	0	0	alu_y
	bne											
J	jal	1	alu_y	0	0	pc	imm(4)	ADD	0	0	1	pc+4(alu_y)
	jalr			rs1		rs1(pc)						
U	lui	-	pc+4	0	0	0	imm	ADD	0	0	1	alu_y
	auipc	-	pc+4			pc						

图表 2：控制信号表

1. add rd, rs1, rs2

   将寄存器 rs1 和 rs2 中的值相加，并将结果写入寄存器 rd

2. and rd, rd1, rs2

   将寄存器 rs1 和 rs2 中的值进行按位与运算，并将结果写入寄存器 rd

3. or rd, rs1, rs2

   将寄存器 rs1 和 rs2 中的值进行按位或运算，并将结果写入寄存器 rd

4. xor rd, rs1, rs2

   将寄存器 rs1 和 rs2 中的值进行按位异或运算，并将结果写入寄存器 rd

5. addi rd, rs1, imm

   将寄存器 rs1 的值与立即数相加，并将结果写入寄存器 rd

6. andi rd, rs1, imm

   将寄存器 rs1 的值与立即数进行按位与运算，并将结果写入寄存器 rd

7. ori rd, rs1, imm

   将寄存器 rs1 的值与立即数进行按位或运算，并将结果写入寄存器 rd

8. slli rd, rs1, imm

   将寄存器 rs1 的值逻辑左移立即数位，并将结果写入寄存器 rd

9. srli rd, rs1, imm

   将寄存器 rs1 的值逻辑右移立即数位，并将结果写入寄存器 rd

10. lb rd, offset(rs1)

    从地址为寄存器 rs1 以偏移量为 offset 读取一个字节，并将其存入寄存器 rd

11. lw rd, offset(rs1)

    从地址为寄存器 rs1 以偏移量为 offset 读取一个字节，并将其存入寄存器 rd

12. sb rs1, offset(rs2)

    将寄存器 rs1 的值存入地址为寄存器 rs2 的增加偏移量的内存，保留低 8 位

13. sw rs1, offset(rs2)

    将寄存器 rs1 的值存入地址为寄存器 rs2 的增加偏移量的内存，保留低 32 位

14. beq rs1, rs2, label

    如果寄存器 rs1 和 rs2 的值相等，则跳转到 label 处执行下一条指令

15. bne rs1, rs2, label

    如果寄存器 rs1 和 rs2 的值不相等，则跳转到 label 处执行下一条指令

16. jal rd, label

    跳转到 label 并将下一条指令地址保存到寄存器 rd 中

17. jalr rd, offset(rs1)

    跳转并将寄存器 rs1 加上偏移量后的地址作为下一条指令地址保存到寄存器 rd

18. lui rd, imm

    将立即数的高 20 位左移立即数位低 12 位清零，并将结果写入寄存器 rd

19. auipc rd, imm

    将立即数的高 20 位和当前指令的地址 pc 相加，将结果写入寄存器 rd

2.1.2 额外指令

1. min rd, rs1, rs2

   将寄存器 rs1 和 rs2 中的值进行比较，将较小的值写入寄存器 rd。

2. xnor rd, rs1, rs2

   将寄存器 rs1 和 rs2 的值进行按位同或，并将结果写入寄存器 rd。

3. clz rd, rs

   统计寄存器 rs 中值的前导零位数并写入寄存器 rd。

2.2 进阶版本：中断异常

2.2.1 内容

实现了运行第二个版本监控程序所需要读写 csr 寄存器所需指令，ecall，ebreak，mret 指令，同时支持时钟中断

2.2.2 实现思路

1. 读写 csr 寄存器都放在 MEM 端，对 load-relate 类数据冲突进行处理
2. 对于 mtime 和 mtimecmp 是以类似于串口的形式实现了单独的 wishbone slave
3. 对于异常的判断是在 exe 段，会将异常信息通过流水线阶段寄存器传至 mem 段，在此段进行异常处理，修改相应的 csr 寄存器

2.2.3 对应模块

- MEM_csrs

  读写 csr 寄存器，进行异常处理

- csr_mtimer

  管理 mtime 和 mtimecmp 寄存器，访问方式和串口相同

第3章　实验展示

3.1　性能测试主频 80Mhz

2:　80001008 <UTEST_1PTB>:35.233s

3:　80001024 <UTEST_2DCT>:19.086s

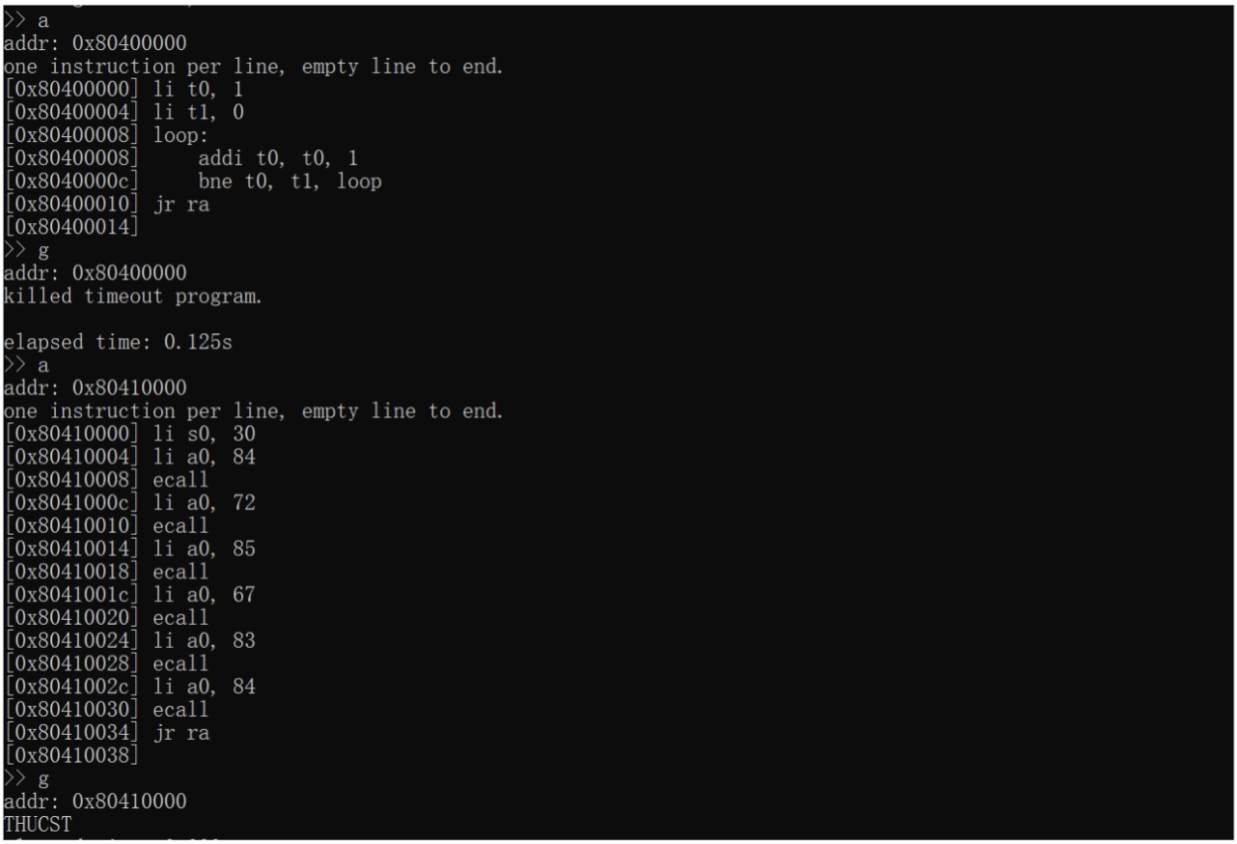
4:　80001064 <UTEST_3CCT>:运行时间过长，超时

5:　80001080 <UTEST_4MDCT>:33.976s

6:　800010a8 <UTEST_CRYPTONIGHT>:1.809s

3.2 中断异常

时钟中断和 ecall 系统调用来输出



ebreak

```
>> a
addr: 0x80420000
one instruction per line, empty line to end.
[0x80420000] li t0,1
[0x80420004] li t1,2
[0x80420008] ebreak
[0x8042000c] li t2,3
[0x80420010] li t3,4
[0x80420014] jr ra
[0x80420018]
>> g
addr: 0x80420000

elapsed time: 0.000s
>> r
R1 (ra)     = 0x8000034c
R2 (sp)     = 0x807f0000
R3 (gp)     = 0x00000000
R4 (tp)     = 0x00000000
R5 (t0)     = 0x00000001
R6 (t1)     = 0x00000002
R7 (t2)     = 0x00000000
R8 (s0/fp)  = 0x0000001e
R9 (s1)     = 0x00000000
R10(a0)     = 0x00000054
R11(a1)     = 0x00000000
R12(a2)     = 0x00000000
R13(a3)     = 0x00000000
R14(a4)     = 0x00000000
R15(a5)     = 0x00000000
R16(a6)     = 0x00000000
R17(a7)     = 0x00000000
R18(s2)     = 0x00000000
R19(s3)     = 0x00000000
R20(s4)     = 0x00000000
R21(s5)     = 0x00000000
R22(s6)     = 0x00000000
R23(s7)     = 0x00000000
R24(s8)     = 0x00000000
R25(s9)     = 0x00000000
R26(s10)    = 0x80420000
R27(s11)    = 0x00000000
R28(t3)     = 0x00000000
R29(t4)     = 0x00000000
R30(t5)     = 0x00000000
R31(t6)     = 0x00000000
```

第4章　思考题目

5.1　第一题：流水线 CPU 设计与多周期 CPU 设计的异同？插入等待周期（气泡）和数据旁路在处理数据冲突的性能上有什么差异 ？

	多周期	流水线
结构	多个阶段（如取指、译码、执行、访存、写回），允许多条指令同时在不同阶段执行	多个周期，每个周期完成一个特定的操作（如取指周期、执行周期、访存周期等），每条指令需要多个周期完成执行
指令执行时间	在流水线中，各个阶段的处理时间可以不同，因此每条指令的执行时间可能会相对较短。	每条指令执行的时间是固定的，根据指令的类型和需要的操作周期数确定。

- **插入等待周期（气泡）：**在流水线中，当出现数据相关或者控制相关的冲突时，可能需要在流水线中插入空操作周期（气泡）来解决冲突。这样做会导致流水线暂停，等待数据准备或者控制信号。

- **数据旁路：**数据旁路是一种技术，允许数据从一个计算单元直接传递到另一个计算单元，而不必经过寄存器文件。这种方式可以减少或避免数据相关带来的延迟，提高流水线的效率。

5.2 第二题：如何使用 Flash 作为外存，如果要求 CPU 在启动时，能够将存放在 Flash 上固定位置的监控程序读入内存，CPU 应当做什么样的改动？

如何使用 Flash 作为外存，如果要求 CPU 在启动时，能够将存放在 Flash 上固定位置的监控程序读入内存，CPU 应当做什么样的改动？

- CPU 需要设计地址空间的映射，且同时由于外设个数改变，需修改 wb_mux
- CPU 还需实现 flash controller，处理 wishbone 中的信号与 flash 交互
- CPU 硬件实现一个 boot loader，在启动时通过总线将 Flash 固定位置的全部内容——监控程序写入内存，最后读入成功后跳转到内存中监控程序对应位置开始运行

5.3 第三题：如何将 DVI 作为系统的输出设备，从而在屏幕上显示文字？

保存好每个字符对应的图像的各个像素点的取值，然后计算屏幕上的相对位置来保证不重叠的同时显示字符图像

5.4 第七题：（异常与中断）假设第 a 个周期在 ID 阶段发生了 Illegal Instruction 异常，你的 CPU 会在周期 b 从中断处理函数的入口开始取指令执行，在你的设计中，b - a 的值为？

待该指令执行至 mem 段，会在 mem 段进行异常处理，下一个周期就会从处理函数如何取指令执行，b-a 的值为 3