流水线 CPU 设计

李卓 pb19000064

实验目的

理解流水线 CPU 的结构和工作原理

掌握流水线 CPU 的设计和调试方法，特别是流水线中数据相关和控制相关的处理

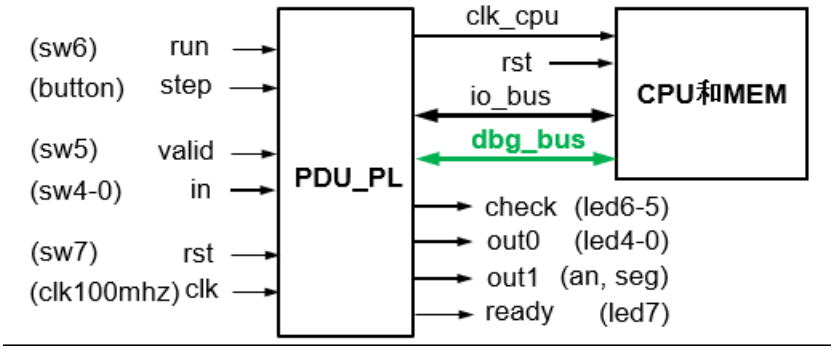
熟练掌握数据通路和控制器的设计和描述方法

实验原理

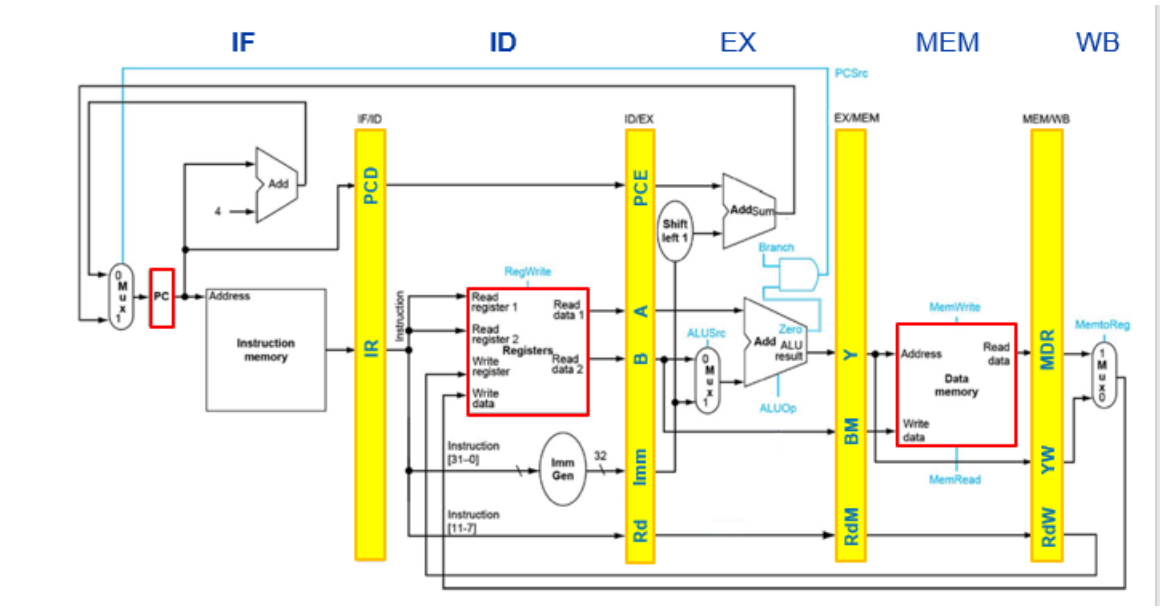
设计实现 5 级流水线的 RISC-V CPU

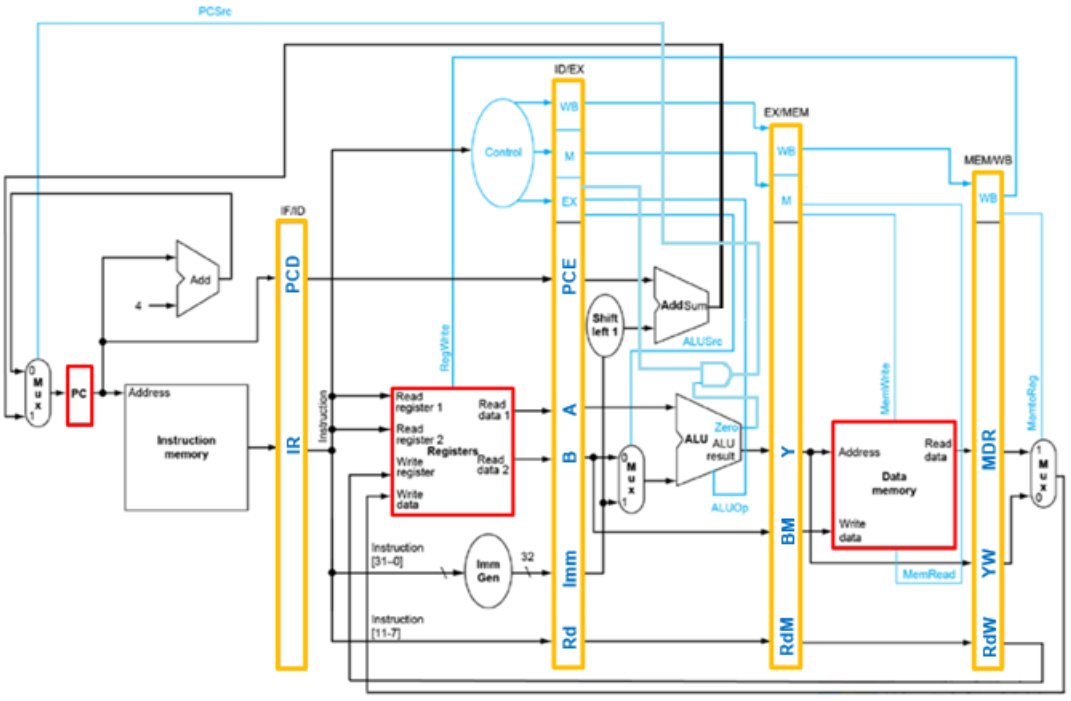
能够执行 6 条指令：**add, addi, lw, sw, beq, jal**

指令存储器和数据存储器均使用分布式存储器，容量均为 256x32 位，数据存储器地址为 0x0000_0000 ~ 0x0000_2ffff，指令存储器地址为 0x0000_3000 ~ 0x0000_3ffc。



流水线 CPU 数据通路





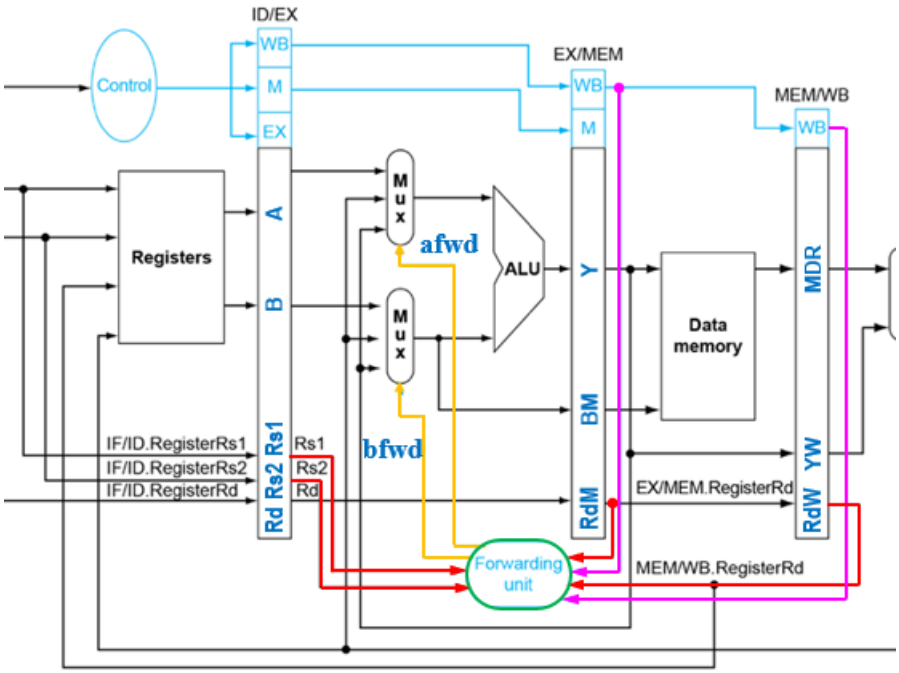
流水线相关及其处理

- 结构相关：当多条指令执行时竞争使用同一资源时
  - 存储器相关处理：哈佛结构（指令和数据存储器分开）
  - 寄存器堆相关处理：同一寄存器读写时，写优先（Write First）
- 数据相关：当一条指令需要等待前面指令的执行结果时
  - 数据定向（Forwarding）：将执行结果提前传递至之前流水段

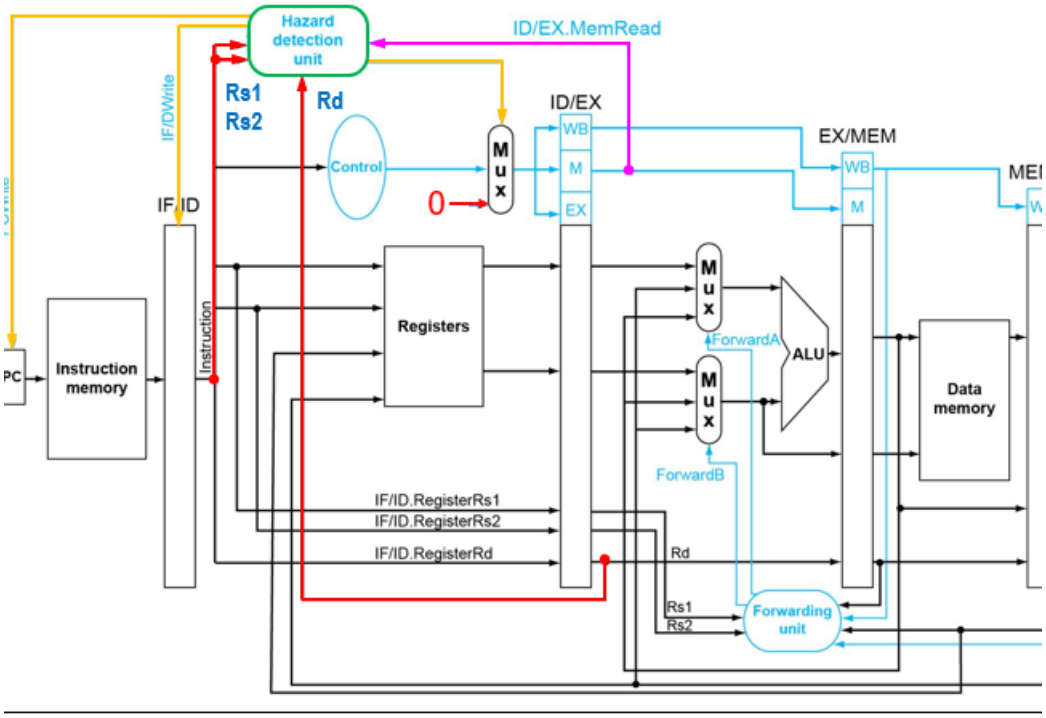
- 加载-使用相关（Load-use hazard）：阻止紧随 Load 已进入流水线的指令流动（Stall），向后续流水段插入空操作（Bubble）

- 控制相关：当遇到转移指令且不能继续顺序执行时
  - 清除（Flush）紧随转移指令已进入流水线的指令 从转移目标处取指令后执行

数据定向（Forwarding）：



load-use hazard:



io_bus 信号

CPU 运行时访问开关(sw)、指示灯(led)和数码管(an, seg)

- io_addr：I/O 外设的地址
- io_din：CPU 接收来自输入缓冲寄存器（IBR）的 sw 输入数据
- io_dout：CPU 向 led 和 seg 输出的数据
- io_we：CPU 向 led 和 seg 输出时的使能信号，利用该信号将 io_dout 存入输出缓冲寄存器（OBR），再经数码管显示电路将其显示在数码管（an，seg）

debug_bus 信号

调试时将存储器和寄存器堆内容，以及 CPU 数据通路状态信息导出显示

- m_rf_addr：存储器(MEM)或寄存器堆(RF)的调试读口地址
- rf_data：从 RF 读取的数据
- m_data：从 MEM 读取的数据
- 流水段寄存器
  - PC/IF/ID：pcin, pc, pcd, ir
  - ID/EX：pce, a, b, imm, rd, ctrl
  - EX/MEM：y, bm, rdm, ctrlm
  - MEM/WB：yw, mdr, rdw, ctrlw

ctrl 信号内容：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
fstall	dstall	dflush	eflush	0	0	a_fwd		0	0	b_fwd		0	rf_wr	wb_sel	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	m_rd	m_wr	0	0	jal	br	0	0	a_sel	b_sel	alu_op			

pdu 运行方式：

sw						btn	led			seg	说明
7	6	5	4~2	1	0		7	6~5	4~0		
rst	run	vld	in			step	rdy	chk	out0	out1	
			ah_m	pre	next						
↑	-	-	-	-	-	-	1	00	0x1F	0x12..78	复位
x	1	vld	in			-	rdy	00	out0	out1	连续运行
	0	vld	in			↑	rdy	00	out0	out1	单步运行
		↑↓	-	↑↓	↑↓	x	rdy	01	a_rf	d_rf	查看寄存器堆
			ah_m					10	al_m	d_m	查看存储器
			-					11	a_plr	d_plr	查看流水段寄存器

查看流水段基寄存器：

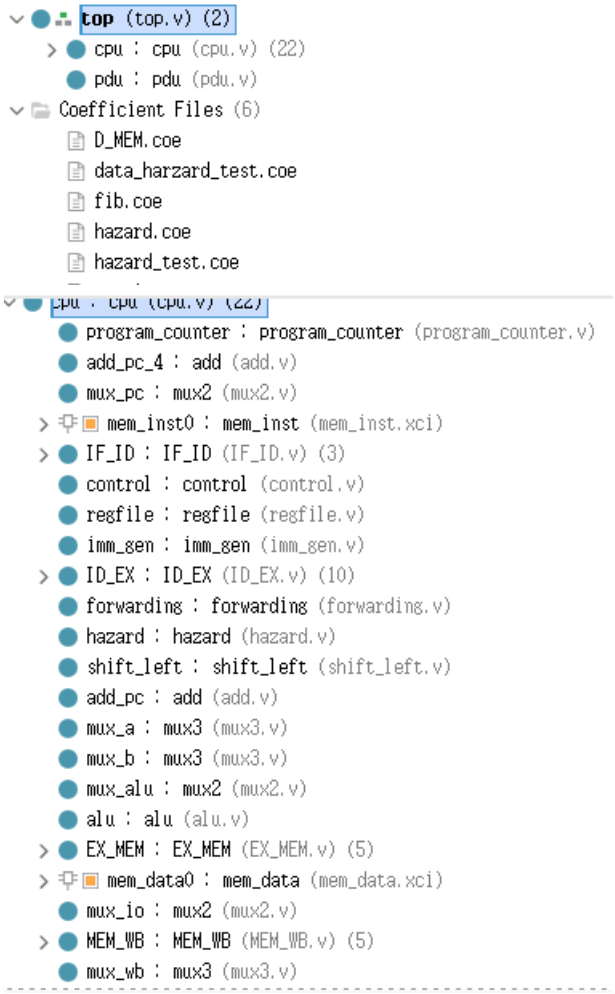
利用 vld (sw5)切换 chk (led6~5) = 11。使用 pre (sw1)和 next (sw0)分别修

改流水段寄存器地址 ah_plr 和 al_plr， out1 (seg)显示对应寄存器数据 (d_plr)

ah_plr (led4~3)	al_plr (led2~0)	d_plr (seg)	ah_plr (led4~3)	al_plr (led2~0)	d_plr (seg)
00 (PC/IF/ID)	x00	pc	10 (EX/MEM)	x00	y
	x01	pcd		x01	bm
	x10	ir		x10	rdm
	x11	pcin		x11	ctrlm
01 (ID/EX)	000	pce	11 (MEM/WB)	x00	yw
	001	a		x01	mdr
	010	b		x10	rdw
	011	imm		x11	ctrlw
	100	rd			
	101	ctrl			

实验过程

源文件结构:



hazard 模块

fstall 阻塞 pc

dstall 阻塞 if-id 流水段

dflush 清空 if-id 流水段

eflush 清空 id-ex 流水段

当 pcsrc 为一时，代表即将发生指令跳转，jal 和 beq 都是在 ex 段判断完成，需要清空 提前进入 if-id 流水段 id-ex 流水段 的指令

当 m_rd 等于一且 rd 与 rs 相等时，代表发生 load-use hazard，阻塞 pc 和 if-id 流水段,并清空 即将 ex 的指令

```
always @(*)
begin
    if(PCSrc)
    begin
        fstall = 1'b0;
        dstall = 1'b0;
        dflush = 1'b1;
        eflush = 1'b1;
    end
    else if(m_rd && rd && (rd==rs1 || rd==rs2))
    begin
        fstall = 1'b1;
        dstall = 1'b1;
        dflush = 1'b0;
        eflush = 1'b1;
    end
    else
    begin
        fstall = 1'b0;
        dstall = 1'b0;
        dflush = 1'b0;
        eflush = 1'b0;
    end

end
```

forwarding 模块:

```
always @(*)
begin
    if (wbm && rdm && rs1 == rdm)
    begin
        forward1 = 2'b01;
    end
    else if (wbw && rdw && rs1 == rdw)
    begin
        forward1 = 2'b10;
    end
    else
    begin
        forward1 = 2'b00;
    end

    if (wbm && rdm && rs2 == rdm)
    begin
        forward2 = 2'b01;
    end
    else if (wbw && rdw && rs2 == rdw)
    begin
        forward2 = 2'b10;
    end
    else
    begin
        forward2 = 2'b00;
    end
end
```

forward1 控制 alu_a 所在的 mux， forward2 控制 alu_b 所在的 mux, forward 为 00 时正常读入，01 时选择从 mem 段来的数据， 10 时选择从 wb 段来的数据

实验代码

所有源文件和测试文件都在仓库中

https://gitee.com/zhuoli628/cod-lab5.git