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

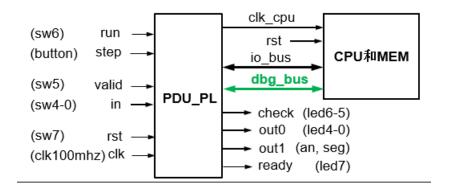
熟练掌握数据通路和控制器的设计和描述方法

实验原理

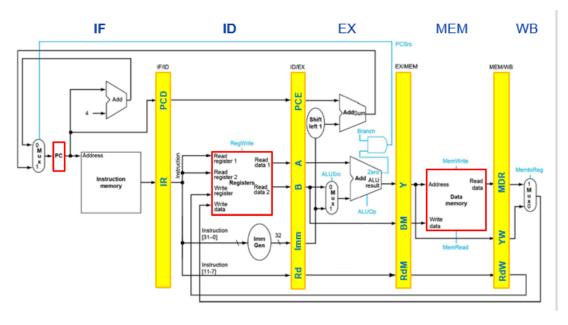
设计实现 5 级流水线的 RISC-V CPU

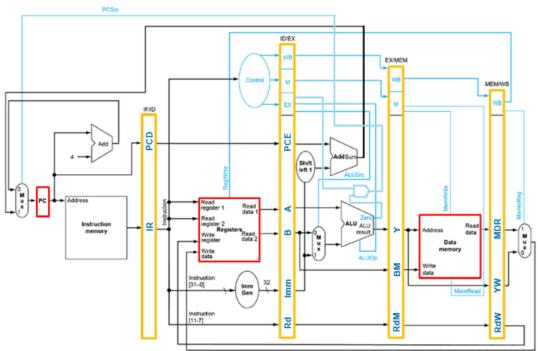
能够执行 6 条指令: add, addi, lw, sw, beq, jal

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)

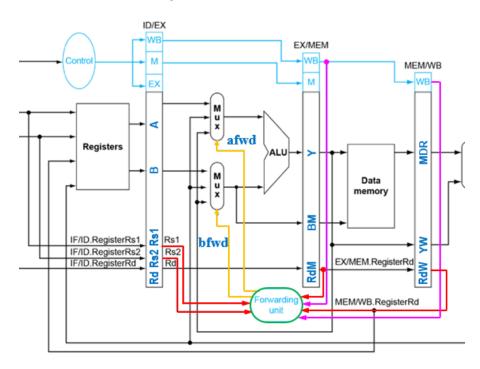
- 寄存器堆相关处理: 同一寄存器读写时, 写优先 (Write First)

• 数据相关: 当一条指令需要等待前面指令的执行结果时

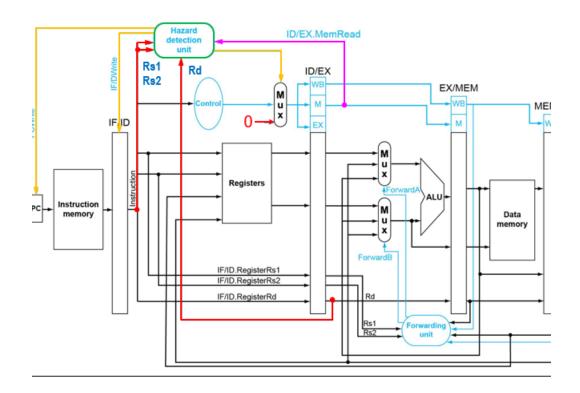
- 数据定向 (Forwarding): 将执行结果提前传递至之前流水段

- 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 f	wd	0	0	b.f	wd	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	ial	br	0	0	a sel	b sel	alu op			

pdu 运行方式:

SW								led		500		
7	6	5	4~2	1	0	<u>btn</u>	7	6~5	4~0	seg	· 说明	
rst	run	vld	in			cton	rd.	chk	out0	out1	版刊	
			ah m	pre	next	step	rdy	CHK	outo	Outi		
1	-	-	-	-	-	-	1	00	0x1F	0x1278	复位	
	1	yld		in		-	rdy	00	out0	out1	连续运行	
х	0	yld	in			1	rdy	00	out0	out1	单步运行	
		↑ ↓	-		†↓	х	rdy	01	a_rf	<u>d.rf.</u>	查看寄存器堆	
			ah.m	↑↓				10	al m	d.m	查看存储器	
			-					11	a plr	<u>d plr</u>	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 x01 x10 x11	pc pcd ir pcin	10 (EX/MEM)	x00 x01 x10 x11	y bm rdm ctrlm
01 (ID/EX)	000 001 010 011 100 101	pce a b imm rd ctrl	11 (MEM/WB)	x00 x01 x10 x11	yw mdr rdw ctrlw

实验过程

源文件结构:

```
v ● ∴ top (top. v) (2)

   > cpu : cpu (cpu.v) (22)
     pdu: pdu (pdu.v)
∨ 🚞 Coefficient Files (6)

□ D_MEM.coe

      data_harzard_test.coe
      📑 fib.coe
      📄 hazard. coe
      hazard_test.coe
v 🌚 բթա - Եթա (Եթա.v) (22)|
     program_counter: program_counter (program_counter.v)
     add_pc_4 : add (add.v)
     mux_pc : mux2 (mux2.v)
   > Ф ■ mem_inst0 : mem_inst (mem_inst.xci)
   > • IF_ID : IF_ID (IF_ID.v) (3)
     control : control (control.v)
     regfile : regfile (regfile.v)
     imm_gen : imm_gen (imm_gen.v)
   > ● ID_EX : ID_EX (ID_EX.v) (10)
     forwarding : forwarding (forwarding.v)
     hazard : hazard (hazard.v)
     shift_left : shift_left (shift_left.v)
     add_pc : add (add.v)
     ■ mux_a : mux3 (mux3.v)
     mux_b : mux3 (mux3.v)
     mux_alu : mux2 (mux2.v)
     ■ alu : alu (alu.v)
   > ● EX_MEM : EX_MEM (EX_MEM. v) (5)
   > 👎 🔳 mem_data0 : mem_data (mem_data.xci)
     mux_io : mux2 (mux2.v)
   > ● MEM_WB : MEM_WB (MEM_WB, v) (5)
     ■ mux_wb : mux3 (mux3.v)
```

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
```

forwarding 模块:

```
always @(*)
begin
    if (wbm && rdm && rs1 == rdm)
    begin
        forward1 = 2'b01;
    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