电子科技大学计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称计算机系统结构综合实验

电子科技大学教务处制表

电 子 科 技 大 学

实　验　报　告

学生姓名：蒋芷昕　　学 号：2017180202005　　指导教师：王华

实验地点：主楼 A2-412　　实验时间：2020.9.13

一、 实验室名称：主楼 A2-412

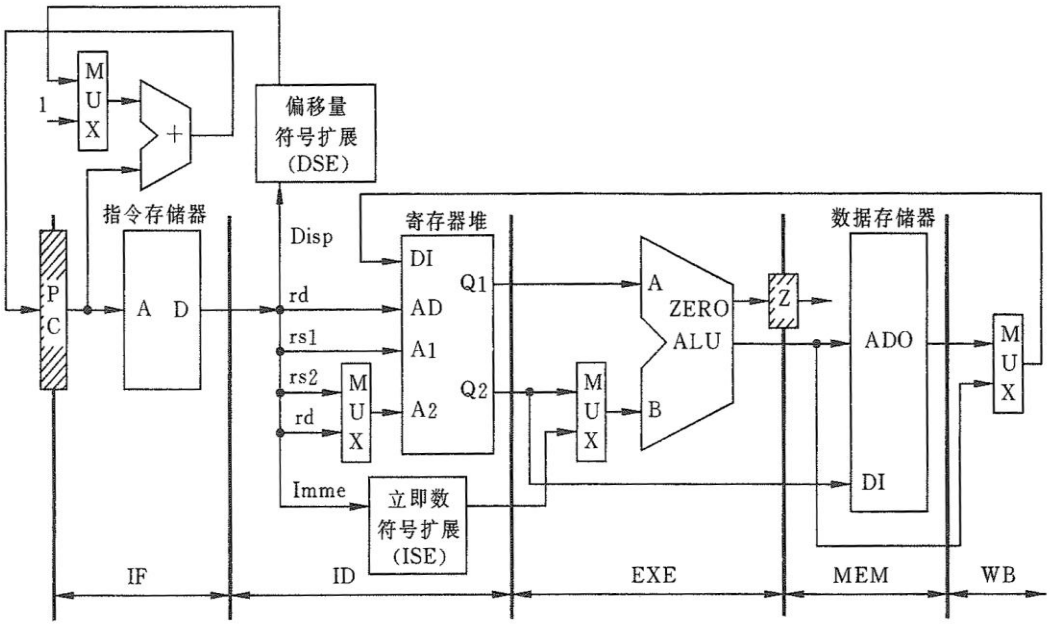
二、 实验项目名称：单周期 CPU 代码分析

三、 实验学时：4 学时

四、 实验原理：（包括知识点，电路图，流程图）

单周期CPU是在一条指令的所有操作全部完成后,才开始下一条指令的执行。

1. 单周期 CPU 电路结构图



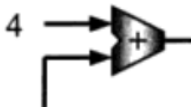
2. 硬件部件

1) 与指令执行有关的电路：

- 指令存储器

- 程序计数器 PC

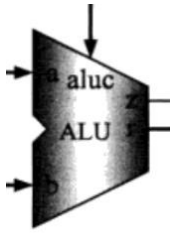
- 修改 PC 值的加法器



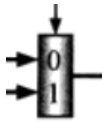
- 选择不同 PC 值的多路选择器

2) 与数据处理有关的电路：

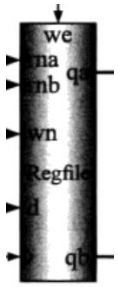
- 算术逻辑运算部件：ALU



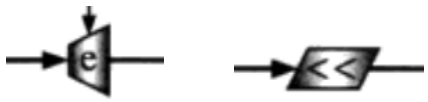
- 与处理方式有关：多路选择器



- 与寄存器有关：寄存器堆



- 与立即数处理有关：数据拓展器、移位器



- 与存储器有关：数据存储器

3. CPU 支持的指令集（32 位）

指令	指令意义	Op[31:26]	Op2 [25:20]	[19:15]	[14:10]	[9:5]	[4:0]
add	寄存器加法	000000	000001	00000	rd	rs	rt
and	寄存器与	000001	000001	00000	rd	rs	rt
or	寄存器或	000001	000010	00000	rd	rs	rt
xor	寄存器异或	000001	000100	00000	rd	rs	rt
srl	逻辑右移	000010	000010	shift	rd	00000	rt
sll	逻辑左移	000010	000011	shift	rd	00000	rt
addi	立即数加法	000101	16 位 immediate			rs	rt
andi	立即数与	001001	16 位 immediate			rs	rt
ori	立即数或	001010	16 位 immediate			rs	rt
xori	立即数异或	001100	16 位 immediate			rs	rt
load	取整数数据字	001101	16 位 offset			rs	rt
store	存整数数据字	001110	16 位 offset			rs	rt
beq	相等则跳转	001111	16 位 offset			rs	rt
bne	不相等则跳转	010000	16 位 offset			rs	rt
jump	无条件跳转	010010	26 位 address				

- 对于 add/and/or/xor rd,rs,rt 指令 //rd←rs op rt
  其中 rs 和 rt 是两个源操作数的寄存器号，rd 是目的寄存器号。
- 对于 sll/srl rd,rt,shift 指令 //rd←rt
  移动 shift 位
- 对于 addi rt,rs,imm 指令 //rt←rs+imm(符号拓展)
  rt 是目的寄存器号，立即数要做符号拓展到 32 位。
- 对于 andi/ori/xori rt,rs,imm 指令 //rt←rs op imm(零拓展) 因为是逻辑指令，所以是零拓展。
- 对于 load rt,offset(rs) 指令 //rt←memory[rs+offset]
  load 是一条取存储器字的指令。寄存器 rs 的内容与符号拓展的 offset 相加得到存储器地址。从存储器取来的数据存入 rt 寄存器。
- 对于 store rt,offset(rs) 指令 // memory[rs+offset]←rt
  store 是一条存字指令。存储器地址的计算方法与 load 相同。
- 对于 beq rs,rt,label 指令 //if(rs==rt) PC←label
- beq 是一条条件转移指令。当寄存器 rs 内容与 rt 相等时，转移到 label。如果程序计数器 PC 是 beq 的指令地址，则 label=PC+4+offset<<2。offset 左移两位导致 PC 的最低两位永远是 0，这是因为 PC 是字节地址，而一条指令要占 4 个字节。offset 要

进行符号拓展，因为 beq 能实现向前和向后两种转移。

- bne 指令与 beq 类似，当寄存器 rs 内容与 rt 不相等时，转移到 label。
- 对于 jump target 指令 //PC←target jump 是一条跳转指令。target 是转移的目标地址，32 位，由 3 部分组成：最高 4 位来自于 PC+4 的高 4 位，中间 26 位是指令中的 address，最低两位为 0。

五、 实验目的：

1. 掌握单周期 CPU 的特点；
2. 熟悉 Verilog HDL 硬件设计语言；
3. 熟悉 Xilinx ISE Design Suite 14.7 集成开发环境。

六、 实验内容：（介绍自己所选的实验内容）

1. 认真阅读并分析所给的单周期 CPU 代码，掌握单周期 CPU 电路结构中各模块的工作原理；
2. 对单周期 CPU 中两个模块进行仿真：
   1） IF_STAGE（取指阶段）
   2） Control_Unit（控制单元）
   分析并理解仿真结果，验证模块逻辑功能；
3. 设计一个指令序列（要求涵盖 CPU 指令集中所有类型的指令，每类指令至少一条）。将指令序列写入指令存储器 inst_mem 中，使用该指令序列对 SCCPU（单周期 CPU 完整电路模块）进行仿真，分析理解仿真结果，掌握单周期 CPU 的工作原理。

七、 实验器材（设备、元器件）：

1. 操作系统：Windows7（64 位）；
2. 开发平台:Xilinx ISE Design Suite 14.7 集成开发系统；
3. 下载软件：digilent.adept.system_v2.10.2.exe（由 FPGA 开用板厂家提供，用于将 Xilinx 开发生成的流代码 bit 文件下载到 FPGA 开发板上）；
4. 编程语言：Verilog HDL 硬件描述语言。

八、 实验步骤：（编辑调试的过程）

1. 分析 IF_STAGE 模块
   输入信号：
   - clk：时钟信号
   - clrn：复位信号，低电平有效，此时 PC 被清零
   - bpc(32 位)：分支指令的下一条地址
   - jpc(32 位)：跳转指令的下一条地址
   - pcsource（2 位）：选择信号，选择下一条指令的地址来源

   输出信号：

- pc4（32 位）：PC+4，用于输出到 ID 级计算下一条指令地址
- inst：执行的指令
- PC（32 位）：指令指针

包含的模块：

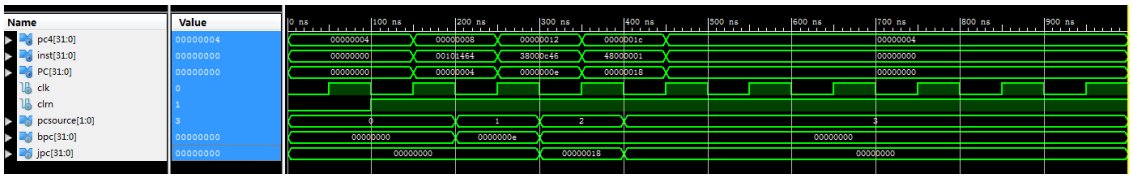
- dff32 program_counter(npc,clk,clrn,pc)：利用 32 位的 D 触发器实现 PC
- add32 pc_plus4(pc,32'h4,pc4)：32 位加法器，用来计算 PC+4
- mux32_4_1 next_pc(pc4,bpc,jpc,32'b0,pcsource,npc)：根据 pcsource 信号选择下一条指令的地址
- Inst_ROM inst_mem(pc[7:2],inst)：指令存储器

2. 对 IF_STAGE 模块仿真

1）编写测试代码

```
initial begin
    // Initialize Inputs
    clk = 0;
    clrn = 0;
    pcsource = 0;//pc+4
    bpc = 0;
    jpc = 0;

    // Wait 100 ns for global reset to fini
    #100;

    // Add stimulus here
    clrn = 1;
    pcsource = 0;//pc+4
    bpc = 0;
    jpc = 0;
    #100;

    clrn = 1;
    pcsource = 1;//bpc分支指令的下一条地址
    bpc = 00000014;//6'h05
    jpc = 0;
    #100;

    clrn = 1;
    pcsource = 2;//jpc分支指令的下一条地址
    bpc = 0;
    jpc = 00000024;//6'h09
    #100;

    clrn = 1;
    pcsource = 3;
    bpc = 0;
    jpc = 0;
    #100;

end
    always #50 clk=~clk;
```

2）仿真



3. 分析 Control_Unit 模块

控制部件负责根据指令的类型，分解指令内容产生各类控制信号。

输入信号：

- rsrtequ：branch 控制信号，判断 ALU 输出结果是否为 0
  if(r=0) rsrtequ=1
- func，op：指令中相应控制码字段

输出信号：

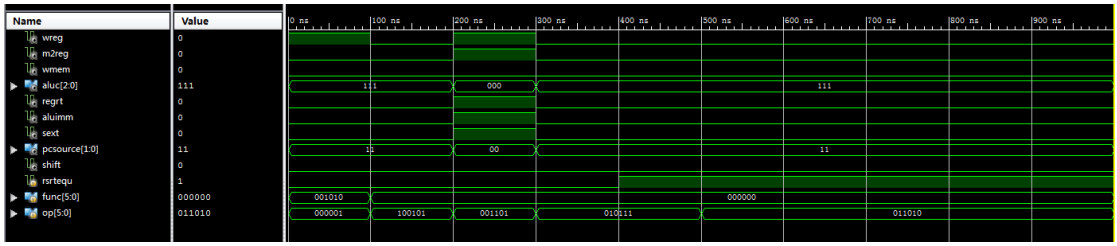
- wreg,m2reg,wmem,regrt,aluimm,sext,shift：cpu 控制信号
- aluc（20 位）：ALU 控制码
- pcsource（2 位）：PC 多路选择器控制码

4. 对 Control_Unit 模块仿真

1) 编写测试代码

```
initial begin
   // Initialize Inputs
   rsrtequ = 0;
   func = 000010;
   op = 000001;//or

   // Wait 100 ns for global
   #100;

   // Add stimulus here
   rsrtequ = 0;
   func = 0;
   op = 000101;//addi
   #100;

   rsrtequ = 0;
   func = 0;
   op = 001101;//load
   #100;

   rsrtequ = 0;
   func = 0;
   op = 001111;//beq
   #100;

   rsrtequ = 1;
   func = 0;
   op = 001111;//beq
   #100;

   rsrtequ = 1;
   func = 0;
   op = 010010;//jump
   #100;

end
```

2) 仿真



5. 设计指令序列

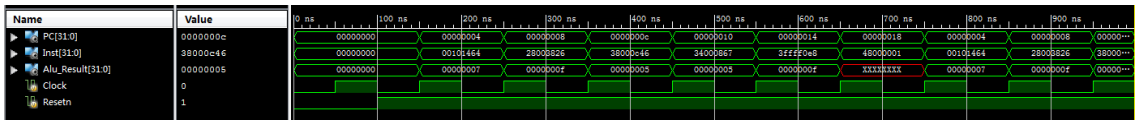
```
assign rom[6'h00]=32'h00000000;    //0地址为空，从1地址开始执行；
assign rom[6'h01]=32'h00101464;    //add r5,r3,r4  r5=0x00000007
assign rom[6'h02]=32'h28003826;    //ori r6,r1,0x000e  r6=0x0000000f
assign rom[6'h03]=32'h38000c46;    //store r6,0x0003(r2)  m5=0x0000000f
assign rom[6'h04]=32'h34000867;    //load r7,0x0002(r3)  r7=0x0000000f
assign rom[6'h05]=32'h3ffff0e8;    //beq r7,r8,6'h02  offset=0xfffc
assign rom[6'h06]=32'h48000001;    //jump 0x0000001
assign rom[6'h07]=32'h041018a1;//and r6,r5,r1  r6=0x00000001
assign rom[6'h08]=32'h04201ca1;//or r7,r5,r1  r7=0x00000007
assign rom[6'h09]=32'h04401461;//xor r5,r3,r1  r5=0x00000002
assign rom[6'h0A]=32'h08211407;//srl r9,r7,0x0002  r9=0x00000001
assign rom[6'h0B]=32'h0831a009;//sll r8,r9,0x0003  r8=0x00000008
assign rom[6'h0C]=32'h240018c7;//andi r5,r6,0x0006  r5=0x00000006
assign rom[6'h0D]=32'h140010a6;//addi r5,r6,0x0004  r5=0x0000000a
assign rom[6'h0E]=32'h300034a8;//xori r5,r8,0x000d  r5=0x00000005
assign rom[6'h0F]=32'h43fff121;//bne r9,r1,6'h02  offset=0xfffc
assign rom[6'h10]=32'h48000001;//jump 0x0000001
```

6. 对 SCCUP 仿真

1) 编写测试代码

```
initial begin
   // Initialize Inputs
   Clock = 0;
   Resetn = 0;

   // Wait 100 ns for global re
   #100;

   // Add stimulus here
   Resetn = 1;

end
   always #50 Clock=~Clock;
```

2) 仿真



九、 实验数据及结果分析：（实验运行结果介绍或者截图，对不同的结果进行分析）

1. IF_STAGE 模块仿真结果分析

将 pcsource 输入信号分别置为 0、1、2、3，验证模块顺序执行、分支执行，以及跳转执行的操作正确性。

1) 当 pcsource=0，复位信号置 0，npc=PC+4，顺序取出地址 1 的指令为 32'h00101464，如图所示：

```
assign rom[6'h01]=32'h00101464;    //add r5,r3,r4  r5=0x00000007
```

此时 inst=32'h00101464，与仿真结果相符。

2) 当 pcsource=1，复位信号置 1，选择 bpc 分支指令的下一条地址的指令执行，假设地址为 32’ h03，如下图所示：

```
assign rom[6'h03]=32'h38000c46;    //store r6,0x0003(r2)  m5=0x0000000f
```

此时 inst=32'h38000c46，与仿真结果相符。

3) 当 pcsource=2，复位信号置 1，选择 jpc 跳转指令的下一条地址的指令执行，假设地址为 32’ h06，如下图所示：

```
assign rom[6'h06]=32'h48000001;    //jump 0x0000001
```

此时 inst=32'h48000001，与仿真结果相符。

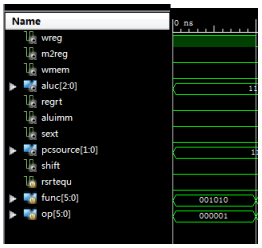
4) 当 pcsource=3，复位信号置 1， npc= 32'b0，根据 Inst_ROM 的初始化，该地址内容为全零，即执行一个空指令，则 inst=0。

```
assign rom[6'h00]=32'h00000000;    //0地址为空，从1地址开始执行；
```

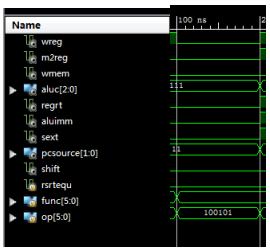
2. Control_Unit 模块结果分析

选择不同类型指令，验证执行过程的正确性。

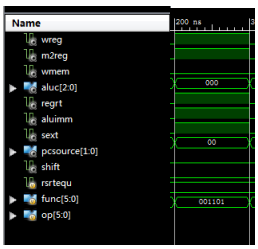
1) Rsrtequ 置 0，选择 or 指令操作码



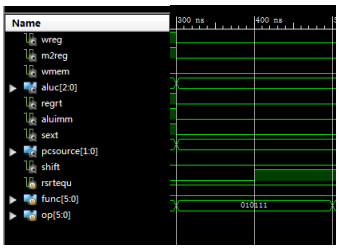
2) Rsrtequ 置 0，选择 addi 指令操作码



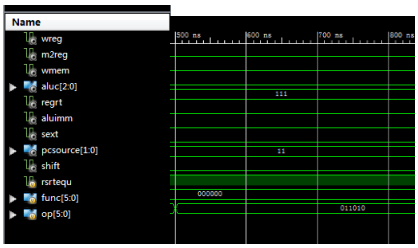
3) Rsrtequ 置 0，选择 load 指令操作码



4) Rsrtequ 置 0 和 1，选择 beq 指令操作码



5) Rsrtequ 置 1，选择 jmp 指令操作码



3. SCCUP 仿真结果分析

Resetn 信号置 1，按照指令序列的初始化内容，依次取出指令执行，执行结果符合预期。

十、 总结及心得体会：（联系理论知识进行说明）

通过本次实验，通过 Verilog HDL 硬件描述语言的实际代码编写，加深了

对单周期 CPU 结构的理解，直观的学习了各个部件的工作原理与流程，巩固了对计算机体系结构的认识和掌握。
根据系统指令集自己设计指令，充盈指令存储器，加深了对指令集，以及操作码的理解，为之后流水线 CPU 的相关操作打下基础。

十一、 对本实验过程及方法、手段的改进建议：

可以尝试为指令寄存器初始化不同的指令集，比较不同指令集的优劣，与设计巧思。

报告评分：

指导教师签字：

实　验　报　告

学生姓名：蒋芷昕　　学 号：2017180202005　　指导教师：王华

实验地点：主楼 A2-412　　实验时间：2020.9.19

一、 实验室名称：主楼 A2-412

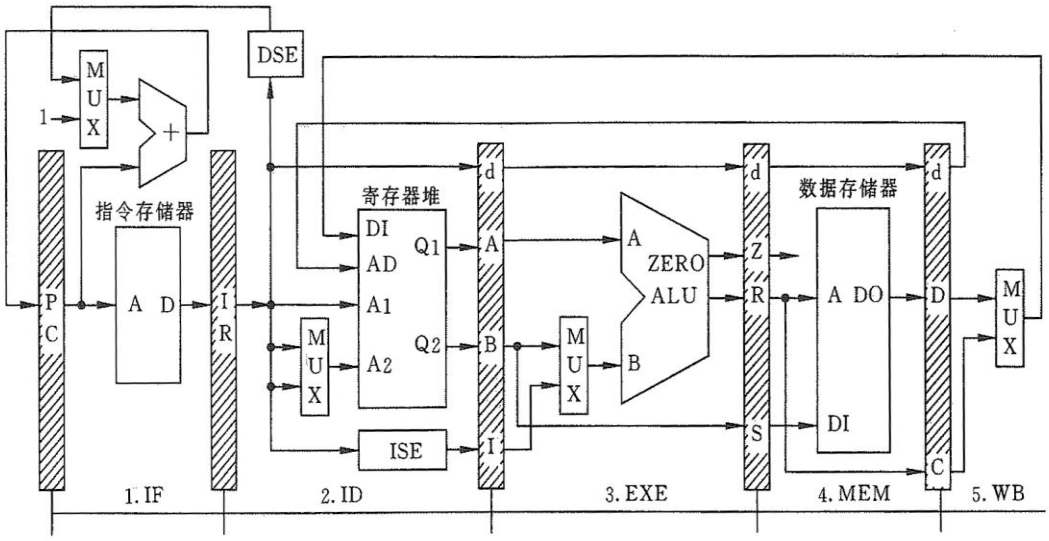
二、 实验项目名称：五级流水线 CPU 实现

三、 实验学时：4 学时

四、 实验原理：（包括知识点，电路图，流程图）

流水线是一种能使多条指令重叠操作的处理机的实现技术。

1. 流水线 CPU 电路结构图



2. 流水线寄存器

必须在流水线的各级之间安排一组寄存器，用来保存当前时钟周期运算出的结果，以便为下个周期使用。注意只能使用触发器寄存器，它将时钟上升沿时数据输入端的信息打入寄存器中；而不能使用锁存器，因为锁存器的输出在时钟高电平时跟随输入的变化而变化。

- IR 寄存器：存放指令
- PC：存放 32 位指令的字地址
- A、B：存放从寄存器堆中读出的两个 32 位数据
- I：存放经符号拓展后的 32 位立即数
- D：保存目的寄存器号
- Z：存在 ALU 的一位 ZERO 标志
- R：保存 32 位 ALU 运算结果
- S：转为 store 指令设置，存放要被写入存储器中断数据

- D：存放 load 指令从存储器中读出的数据
- C：保存前一级的 R，即 ALU 运算结果。D 和 C 的数据要写入由 d 指定的目的寄存器中

3. 流水线级
   1) IF 级
   第一级为取指令级。处理及使用 PC 的内容访问指令存储器，取出指令，并在该级结束时，将指令打入寄存器。下一条指令的地址也在该级计算出，并将它打入 PC 寄存器。
   2) ID 级
   第二级为指令译码级。数据路径需要从寄存器堆中读寄存器操作数额对指令中的立即数部分进行符号拓展；控制部件根据指令操作码 OPCODE，产生所有控制信号。
   3) EXE 级
   第三级为执行级。ALU 运行类型的指令将在本级 ALU 计算出结果，并将其打入寄存器；ALU 的 ZERO 输出被打入寄存器。
   4) MEM 级
   第四级为存储器访问级，专为 LOAD\STORE 指令设置。
   5) WB 级
   第五级为写回级，将指令结果写回到寄存器堆。

五、 实验目的：

1. 掌握流水线 CPU 和单周期 CPU 的区别；
2. 进一步熟悉 Verilog HDL 硬件设计语言；
3. 进一步掌握开发平台 Xilinx ISE Design Suite 14.7 集成开发系统的操作方法。

六、 实验内容：（介绍自己所选的实验内容）

1. 在单周期 CPU 代码的基础上添加流水线，对以下文件补充代码，以构建具有五级流水线结构的 CPU：
   1) IF_ID 级流水线寄存器（instruction_register）
   2) ID_EXE 级流水线寄存器（id_exe_register）
   3) EXE_MEM 级流水线寄存器（exe_mem_register）
   4) MEM_WB 级流水线寄存器（mem_wb_register
2. 按以下方式对寄存器与存储器进行初始化：
   - 寄存器

```
register[5'h01]<=32'h00000001;
register[5'h02]<=32'h00000002;
register[5'h03]<=32'h00000003;
register[5'h04]<=32'h00000004;
register[5'h05]<=32'h00000005;
register[5'h06]<=32'h00000006;
register[5'h07]<=32'h00000007;
```

register[5'h08]<=32'h00000008;

- 存储器

ram[5'h01]=32'h00000001;
ram[5'h02]=32'h00000002;
ram[5'h03]=32'h00000003;
ram[5'h04]=32'h00000004;
ram[5'h05]=32'h00000005;
ram[5'h06]=32'h00000006;
ram[5'h07]=32'h00000007;
ram[5'h08]=32'h00000008;

3. 对所实现的流水线 CPU 进行仿真，验证以下指令序列的运行结果，对仿真结果进行分析：

```
addi  r1, r1, 0x0004
load  r2, 0x0004(r3)
or  r4, r5, r6
add  r3, r5, r6
store  r8, 0x0002(r7)
srl  r9, r7, 0x02
```

4. 分析掌握流水线 CPU 电路结构与工作原理，画出具有五级流水线的 CPU 电路结构图：
   1) 包含每级流水线中的主要功能模块；
   2) 标出 CPU 结构中各信号名称及传递方向；
   3) 简要说明各信号在 CPU 工作流程中的作用。

七、 实验器材（设备、元器件）：

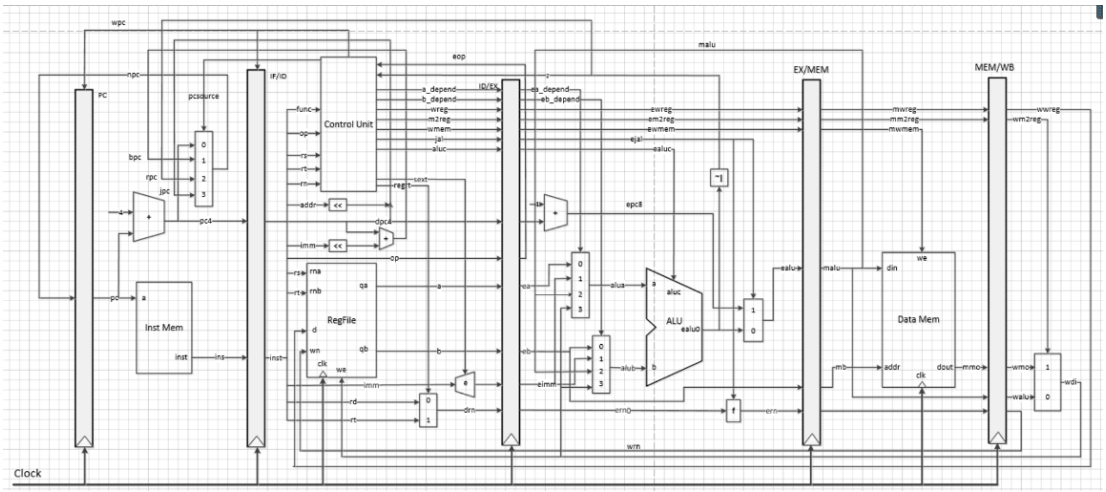
- 操作系统：Windows7（64 位）；
- 开发平台:Xilinx ISE Design Suite 14.7 集成开发系统；
- 下载软件：digilent.adept.system_v2.10.2.exe（由 FPGA 开用板厂家提供,用于将 Xilinx 开发生成的流代码 bit 文件下载到 FPGA 开发板上）；
- 编程语言：Verilog HDL 硬件描述语言。

八、 实验步骤：（编辑调试的过程）

1. 补全具有五级流水线的电路结构图，分析模块功能与信号传递；
2. 在单周期 CPU 代码的基础上添加流水线寄存器，补充代码，构建具有五级流水线结构的 CPU；
3. 初始化寄存器和存储器；
4. 对流水线 CPU 仿真。

九、 实验数据及结果分析：（实验运行结果介绍或者截图，对不同的结果进行分析）

1. 具有五级流水线的电路结构图



2. 添加流水线寄存器

a） IR 寄存器

```
instruction_register IR(if_pc4, IF_Inst, Clock, Resetn, id_pc4, ID_Inst);


 //请在下方补充代码以完成流水线寄存器
 ////////////////////////////////////////.
 always @(negedge clrn or posedge clk)
 if(clrn==0)
   begin
      id_pc4<=0;
      id_inst<=0;
   end
else
   begin
      id_pc4<=if_pc4;
      id_inst<=if_inst;
   end
```

b） ID _EXE 级寄存器

```
id_exe_register ID_EXE (.clk(Clock), .clrn(Resetn),
                        .id_wreg(id_wreg), .id_m2reg(id_m2reg), .id_wmem(id_wmem), .id_aluc(id_aluc), .id_aluimm(id_aluimm),
                        .id_a(id_a), .id_b(id_b), .id_imm(id_imm), .id_rn(id_rn), .id_shift(id_shift), .id_wz(id_wz),
                        .exe_wreg(exe_wreg), .exe_m2reg(exe_m2reg), .exe_wmem(exe_wmem), .exe_aluc(exe_aluc), .exe_aluimm(exe_aluimm),
                        .exe_a(exe_a), .exe_b(exe_b), .exe_imm(exe_imm), .exe_rn(exe_rn), .exe_shift(exe_shift), .exe_wz(exe_wz));
```

```
//请在下方补充代码以完成流水线寄存器
//////////////////////////////////////////////////////////////////
reg [31:0] exe_a,exe_b,exe_imm;
reg [4:0] exe_rn;
reg [2:0] exe_aluc;
reg eexe_wreg,exe_m2reg,exe_wmem,exe_aluimm,exe_shift,exe_wz;
always @(negedge clrn or posedge clk)
    if(clrn==0)
        begin
            exe_wreg<=0;
            exe_m2reg<=0;
            exe_wmem<=0;
            exe_aluc<=0;
            exe_aluimm<=0;
            exe_a<=0;
            exe_b<=0;
            exe_imm<=0;
            exe_rn<=0;
            exe_shift<=0;
            exe_wz<=0;
        end
     else
        begin
            exe_wreg<=id_wreg;
            exe_m2reg<=id_m2reg;
            exe_wmem<=id_wmem;
            exe_aluc<=id_aluc;
            exe_aluimm<=id_aluimm;
            exe_a<=id_a;
            exe_b<=id_b;
            exe_imm<=id_imm;
            exe_rn<=id_rn;
            exe_shift<=id_shift;
            exe_wz<=id_wz;
```

c） EXE_MEM 寄存器

```
exe_mem_register_withWZ EXE_MEM (.clk(Clock),.clrn(Resetn),
                                 .exe_z(exe_z),.exe_wz(exe_wz),
                                 .exe_wreg(exe_wreg),.exe_m2reg(exe_m2reg),.exe_wmem(exe_wmem),.exe_alu(EXE_Alu),.exe_b(exe_b),.exe_rn(exe_rn),
                                 .mem_wreg(mem_wreg),.mem_m2reg(mem_m2reg),.mem_wmem(mem_wmem),.mem_alu(MEM_Alu),.mem_b(mem_b),.mem_rn(mem_rn),.mem_z(mem_z));
```

```
//请在下方补充代码以完成流水线寄存器
//////////////////////////////////////////
reg [31:0] mem_alu,mem_b;
reg [4:0] mem_rn;
reg mem_wreg,mem_m2reg,mem_wmem;
always @(negedge clrn or posedge clk)
  if(clrn==0)
     begin
        mem_alu<=0;
        mem_b<=0;
        mem_rn<=0;
        mem_wreg<=0;
        mem_m2reg<=0;
        mem_wmem<=0;
     end
  else
     begin
        mem_alu<=exe_alu;
        mem_b<=exe_b;
        mem_rn<=exe_rn;
        mem_wreg<=exe_wreg;
        mem_m2reg<=exe_m2reg;
        mem_wmem<=exe_wmem;
     end
```

d） MEM_WB 寄存器

```
mem_wb_register MEM_WB (mem_wreg, mem_m2reg, mem_mo, MEM_Alu, mem_rn, Clock, Resetn,
                        wb_wreg, wb_m2reg, wb_mo, WB_Alu, wb_rn);
```

```
//请在下方补充代码以完成流水线寄存器
////////////////////////////////////////
reg [31:0] wb_alu,wb_mo;
reg [4:0] wb_rn;
reg wb_wreg,wb_m2reg;

always @(negedge clrn or posedge clk)
  if(clrn==0)
     begin
        wb_wreg<=0;
        wb_m2reg<=0;
        wb_mo<=0;
        wb_alu<=0;
        wb_rn<=0;
     end
  else
     begin
        wb_wreg<=mem_wreg;
        wb_m2reg<=mem_m2reg;
        wb_mo<=mem_mo;
        wb_alu<=mem_alu;
        wb_rn<=mem_rn;
     end
```

3. 初始化寄存器和存储器

```
module Regfile(rna,rnb,d,wn,we,clk,clrn,qa,qb
    );
    input [4:0] rna,rnb,wn;
    input [31:0] d;
    input we,clk,clrn;
    output [31:0] qa,qb;
    reg [31:0] register [1:31];
    assign qa=(rna==0)?0:register[rna];
    assign qb=(rnb==0)?0:register[rnb];
    always @(posedge clk or negedge clrn)
        if(clrn==0)      //如果复位信号有效，则进行寄
        begin:init
        integer i;
        for(i=1;i<32;i=i+1)
            register[i]<=0;
        //初始化寄存器
        register[5'h01]<=32'h00000001;
        register[5'h02]<=32'h00000002;
        register[5'h03]<=32'h00000003;
        register[5'h04]<=32'h00000004;
        register[5'h05]<=32'h00000005;
        register[5'h06]<=32'h00000006;
        register[5'h07]<=32'h00000007;
        end
        else if((wn!=0)&&we)
           register[wn]<=d;
endmodule
```

```
////////////////////////////////////////////
module memory(we,addr,datain,clk,dataout
    );
    input [31:0] datain;
    input [4:0] addr;
    input clk,we;
    output [31:0] dataout;
    reg [31:0] ram [0:31];
    assign dataout=ram[addr];    //读出常有效
    always @(posedge clk)begin
    if(we)ram[addr]=datain;
    end

    integer i;
    initial begin    //存储器初始化
    for(i=0;i<32;i=i+1)
       ram[i]=0;
    ram[5'h01]=32'h00000001;
    ram[5'h02]=32'h00000002;
    ram[5'h03]=32'h00000003;
    ram[5'h04]=32'h00000004;
    ram[5'h05]=32'h00000005;
    ram[5'h06]=32'h00000006;
    ram[5'h07]=32'h00000007;
    ram[5'h08]=32'h00000008;

    end
endmodule
```

4. 仿真

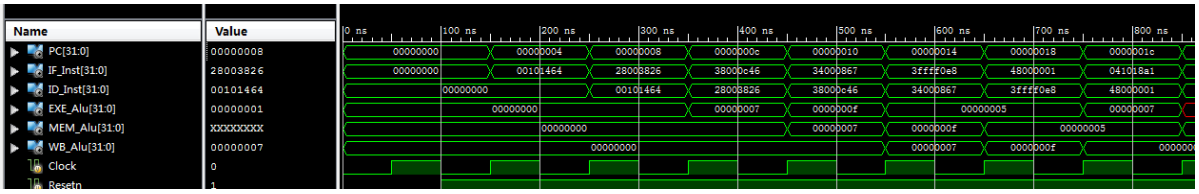
仿真运行的指令序列如下：

```
assign rom[6'h00]=32'h00000000;    //0地址为空，从1地址开始执行；
assign rom[6'h01]=32'h00101464;    //add r5,r3,r4  r5=0x00000007
assign rom[6'h02]=32'h28003826;    //ori r6,r1,0x000e  r6=0x0000000f
assign rom[6'h03]=32'h38000c46;    //store r6,0x0003(r2)  m5=0x0000000f
assign rom[6'h04]=32'h34000867;    //load r7,0x0002(r3)  r7=0x0000000f
assign rom[6'h05]=32'h3ffff0e8;    //beq r7,r8,6'h02  offset=0xfffc
assign rom[6'h06]=32'h48000001;    //jump 0x0000001
assign rom[6'h07]=32'h041018a1;//and r6,r5,r1  r6=0x00000001
assign rom[6'h08]=32'h04201ca1;//or r7,r5,r1  r7=0x00000007
assign rom[6'h09]=32'h04401461;//xor r5,r3,r1  r5=0x00000002
assign rom[6'h0A]=32'h08211407;//srl r9,r7,0x0002  r9=0x00000001
assign rom[6'h0B]=32'h0831a009;//sll r8,r9,0x0003  r8=0x00000008
assign rom[6'h0C]=32'h240018c7;//andi r5,r6,0x0006  r5=0x00000006
assign rom[6'h0D]=32'h140010a6;//addi r5,r6,0x0004  r5=0x0000000a
assign rom[6'h0E]=32'h300034a8;//xori r5,r8,0x000d  r5=0x00000005
assign rom[6'h0F]=32'h43fff121;//bne r9,r1,6'h02  offset=0xfffc
assign rom[6'h10]=32'h48000001;//jump 0x0000001
assign rom[6'h11]=32'h00000000;
```

编写测试代码：

```
    initial begin
        // Initialize Inputs
        Clock = 0;
        Resetn = 0;

        // Wait 100 ns for global res
        #100;

        // Add stimulus here
        Resetn = 1;
    end
        always #50 Clock=~Clock;
endmodule
```

仿真结果如下：



仿真结果符合预期，指令重叠执行，流水线中的每个步骤完成指令的一部分，依次取出指令，从一端进入，通过流水线后在另一端退出。

十、 总结及心得体会：（联系理论知识进行说明）

通过本次实验，实现了五级流水线 CPU ，加深对流水线技术和流水线处理机的理解。

通过实验一和实验二的比较，更为直观地感受到指令重叠技术对于单周期处理器的重要性。同非流水线相比，尽管单条指令的执行时间并没有缩短，但从整体来看，每个时钟周期都会有一条指令执行完毕，极大地提高了系统的吞吐量。

十一、 对本实验过程及方法、手段的改进建议：

无

报告评分：

指导教师签字：

实　验　报　告

学生姓名：蒋芷昕　　学 号：2017180202005　　指导教师：王华

实验地点：主楼 A2-412　　实验时间：2020.9.20

一、 实验室名称：主楼 A2-412

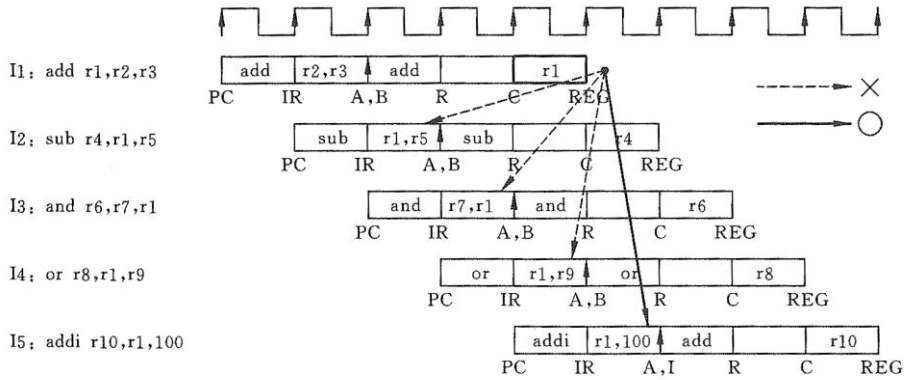
二、 实验项目名称：解决数据冒险问题

三、 实验学时：4 学时

四、 实验原理：（包括知识点，电路图，流程图）

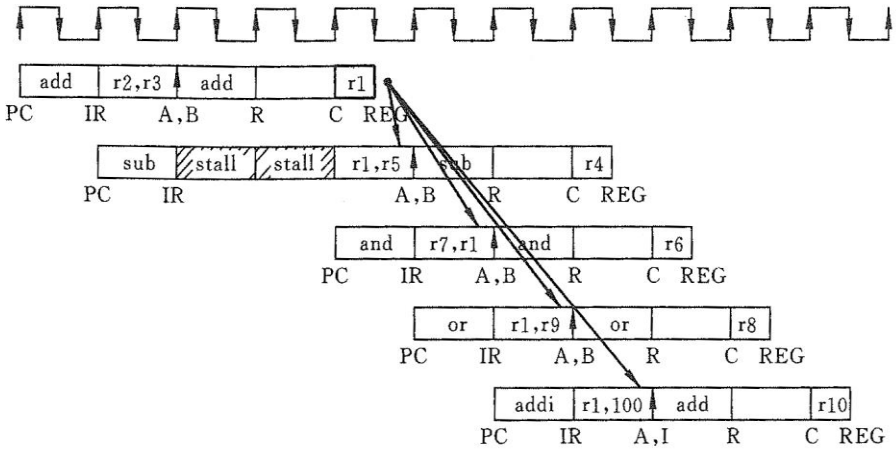
1. 数据冒险

流水线的主要效果是通过重叠指令的执行过程，改变它们的相对执行时间。根据流水线中的指令重叠，指令之间存在先后顺序，如果一条指令取决于先前指令的结构，就可能导致数据冒险。考虑以下指令的流水化执行：



Add 指令之后的所有指令都用到了 add 指令的结果。Add 指令在 WB 级写入 R1 的值，但 sub 指令在其 ID 级中读取这个值。除非提前防范这种问题，否则 sub 指令将会读取错误值并试图使用它。

2. 暂停流水线



3. 内部前推

可以通过硬件技术解决数据冒险问题。在上例中，add 指令将结果放在流水线寄存器中，如果可以把它从这里转移到 sub 需要的地方，就可以避免出现停顿。内部前推的工作方式如下所述。

a) 来着 EX/MEM 和 MEM/WB 流水线寄存器的 ALU 结果总是被反馈回 ALU 的输入端；

b) 如果转发硬件检测到前一个 ALU 操作已经对当前 ALU 操作的源寄存器进行了写入操作，则控制逻辑选择内部前推结果作为 ALU 输入，而不是选择从寄存器堆中读取的值。

五、 实验目的：

1. 掌握流水线 CPU 和单周期 CPU 的区别；
2. 进一步熟悉 Verilog HDL 硬件设计语言；
3. 熟悉和掌握开发平台 Xilinx ISE Design Suite 14.7 集成开发系统的操作方法；
4. 进一步理解和掌握流水线数据冒险的概念和解决方法。

六、 实验内容：（介绍自己所选的实验内容）

1. 使用暂停使用暂停流水线方法解决数据冒险问题：
   1) 补充相关模块代码，设计实现数据冒险检测模块；
   2) 补充相关模块代码，设计实现流水线寄存器暂停功能；
   3) 修改流水线 CPU 代码，当检测到数据冒险时暂停流水线，直至冒险消除时恢复流水线运行。
2. 使用内部前推技术+暂停流水线方法解决数据冒险问题：
   1) 分析数据冒险检测模块；
   2) 分析流水线 CPU 中内部数据前推通路；
   3) 分析流水线 CPU 代码并仿真，分析产生数据冒险时通过内部前推数据通路如何得到正确结果；
   4) 分析当检测到 Load 指令数据冒险时通过内部前推数据+暂停流水线

如何得到正确的计算结果。

3. 对以下指令序列进行仿真，验证所实现流水线 CPU 能够解决数据冒险问题：

```
add  r1, r2, r3;
and  r4, r1, r5;
or  r6, r7, r1;
addi  r8, r1, 0x000a;
load  r1, 0xfff5(r8);
sll  r9, r1, 0x02;
store  r9, 0x0027(r1);
```

（注：可自行设计含有数据冒险的指令序列进行验证。）

4. 对相应寄存器与存储器进行初始化（可参考可以参考实验二的要求，也可以根据自己设计的指令序列更改初始化数据）。
5. 思考：通过内部前推技术，是否能够解决以下指令序列的流水线暂停？若不能，流水线应该如何扩展？

```
add  r3, r1, r4;
store  r3, 200(r2);
```

七、 实验器材（设备、元器件）：

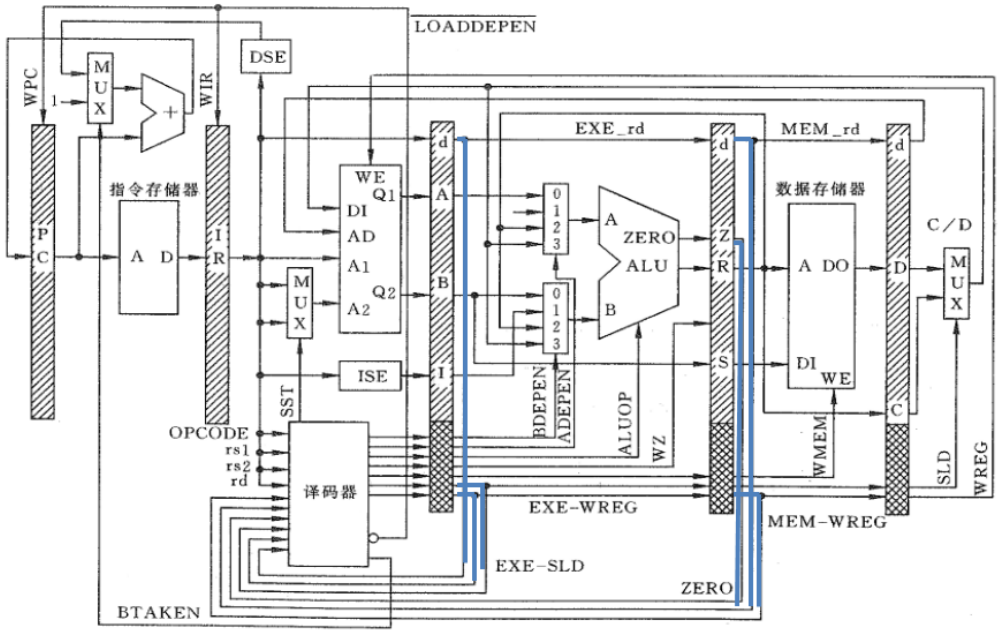
- 操作系统：Windows7（64 位）；
- 开发平台:Xilinx ISE Design Suite 14.7 集成开发系统；
- 下载软件：digilent.adept.system_v2.10.2.exe（由 FPGA 开用板厂家提供,用于将 Xilinx 开发生成的流代码 bit 文件下载到 FPGA 开发板上）；
- 编程语言：Verilog HDL 硬件描述语言。

八、 实验步骤：（编辑调试的过程）

1. 补全数据冒险检测模块；
2. 补全流水线寄存器暂停功能；
3. 修改代码，实现数据冒险的检测和流水线恢复；
4. 初始化寄存器和存储器；
5. 分析流水线 CPU 中内部数据前推通路并仿真；
6. 验证具有数据冒险的指令序列。

九、 实验数据及结果分析：（实验运行结果介绍或者截图，对不同的结果进行分析）

1. 电路结构图



2. 在顶层模块增加信号

```
//增加冲突判断信号
wire [1:0] ID_A_DEPEN,ID_B_DEPEN;
wire [1:0] EXE_A_DEPEN,EXE_B_DEPEN;
wire stall;


program_counter PCR (Clock, Resetn, npc, PC,stall);

instruction_fetch IF_STAGE (pcsource, PC, bpc, jpc, if_pc4, npc, IF_Inst,stall);

instruction_register IR(if_pc4, IF_Inst, Clock, Resetn, id_pc4, ID_Inst);

instruction_decode ID_STAGE (id_pc4, ID_Inst, wdi, Clock, Resetn, bpc, jpc, pcsource,
                             id_m2reg, id_wmem, id_aluc, id_aluimm, id_a, id_b, id_imm,
                             id_shift, mem_z , id_wreg, wb_wreg, id_rn, wb_rn, id_wz,
                             ID_A_DEPEN,ID_B_DEPEN,stall,exe_wreg, mem_wreg,exe_rn, mem_rn,id_store,exe_store );

id_exe_register ID_EXE (.clk(Clock), .clrn(Resetn),
                        .id_wreg(id_wreg), .id_m2reg(id_m2reg), .id_wmem(id_wmem), .id_aluc(id_aluc), .id_aluimm(id_aluimm),
                        .id_a(id_a), .id_b(id_b), .id_imm(id_imm), .id_rn(id_rn), .id_shift(id_shift), .id_wz(id_wz),
                        .exe_wreg(exe_wreg), .exe_m2reg(exe_m2reg), .exe_wmem(exe_wmem), .exe_aluc(exe_aluc), .exe_aluimm(exe_aluimm),
                        .exe_a(exe_a), .exe_b(exe_b), .exe_imm(exe_imm), .exe_rn(exe_rn), .exe_shift(exe_shift), .exe_wz(exe_wz),
                        .EXE_A_DEPEN(EXE_A_DEPEN),.EXE_B_DEPEN(EXE_B_DEPEN));

execute EXE_STAGE (exe_aluc, exe_aluimm, exe_a, exe_b, exe_imm, exe_shift, EXE_Alu, exe_z,
                   EXE_A_DEPEN,EXE_B_DEPEN, MEM_Alu, WB_Alu);
```

3. 修改 PCR

```
always @ (negedge clrn or posedge clk)
if(clrn==0)
    begin
        q<=0;
    end
else
    begin
        if(stall==1)
           begin
              q<=q;
           end
        else
           begin
              q<=d;
           end
    end
end
```

4. 修改 ID_EXE 寄存器

```
always @(negedge clrn or posedge clk)
    if(clrn==0)
       begin
           exe_wreg<=0;
           exe_m2reg<=0;
           exe_wmem<=0;
           exe_aluc<=0;
           exe_a<=0;
           exe_b<=0;
           exe_imm<=0;
           exe_rn<=0;
           exe_wz<=0;
           EXE_A_DEPEN<=2'h0;
           EXE_B_DEPEN<=2'h0;
           exe_store<=0;
       end
     else
       begin
           exe_wreg<=id_wreg;
           exe_m2reg<=id_m2reg;
           exe_wmem<=id_wmem;
           exe_aluc<=id_aluc;
           exe_a<=id_a;
           exe_b<=id_b;
           exe_imm<=id_imm;
           exe_rn<=id_rn;
           exe_wz<=id_wz;
           EXE_A_DEPEN<=ID_A_DEPEN;
           EXE_B_DEPEN<=ID_B_DEPEN;
           exe_store<=id_store;
       end
```

5. 增加控制信号

```
Control_Unit cu(rsrtequ,func,      //控制部件
            op,id_wreg,m2reg,wmem,aluc,regrt,
            sext,pcsource,id_wz,
            ID_A_DEPEN,ID_B_DEPEN,stall,exe_wreg, mem_wreg,id_store,exe_store,
            exe_rn==rs,mem_rn==rs,exe_rn==rt,mem_rn==rt,exe_rn==rd,mem_rn==rd);


////////////////////////////////////////////控制信号的生成//////////////////////////
//判断rs1是否为寄存器操作数
assign rs1IsReg=i_and|i_andi|i_or|i_ori|i_add|i_addi|i_xor|i_xori|i_lw|i_sw;
//判断rs2是否为寄存器操作数
assign rs2IsReg=i_and|i_or|i_xor|i_add|i_srl|i_sll;
//计算A_DEPEN
assign DEPEN=A_DEPEN|B_DEPEN;
assign A_DEPEN=EXE_A_DEPEN|MEM_A_DEPEN;
assign EXE_A_DEPEN=exe_equ_rs&exe_wreg&rs1IsReg;
assign MEM_A_DEPEN=mem_equ_rs&mem_wreg&rs1IsReg;
//计算B_DEPEN
assign B_DEPEN=EXE_B_DEPEN|MEM_B_DEPEN;
assign EXE_B_DEPEN=(exe_equ_rt&exe_wreg&rs2IsReg)|(exe_equ_rd&exe_wreg&i_sw);
assign MEM_B_DEPEN=(mem_equ_rt&mem_wreg&rs2IsReg)|(mem_equ_rd&mem_wreg&i_sw);
//计算ID_A_DEPEN和ID_B_DEPEN
assign ID_A_DEPEN[0]=MEM_A_DEPEN;
assign ID_A_DEPEN[1]=MEM_A_DEPEN|EXE_A_DEPEN;
assign ID_B_DEPEN[0]=MEM_B_DEPEN|(!rs2IsReg);
assign ID_B_DEPEN[1]=MEM_B_DEPEN|EXE_B_DEPEN;
//判断load指令冒险，若stall=0则stall一个时钟周期
assign stall=(exe_equ_rs&exe_store&rs1IsReg)|(exe_equ_rt&exe_store&rs2IsReg);
assign id_store=i_sw;
```

6. 修改 4 选 1 多路选择器

```
//变为4选1多路选择
mux32_4_1 alu_ina (exe_a,sa,MEM_Alu,WB_Alu,EXE_A_DEPEN,alua);//选择ALU a端的数据来源
mux32_4_1 alu_inb (exe_b,exe_imm,MEM_Alu,WB_Alu,EXE_B_DEPEN,alub);//选择ALU b端的数据来源
alu al_unit (alua,alub,exe_aluc,exe_alu,z);//ALU
```

7. 修改指令序列

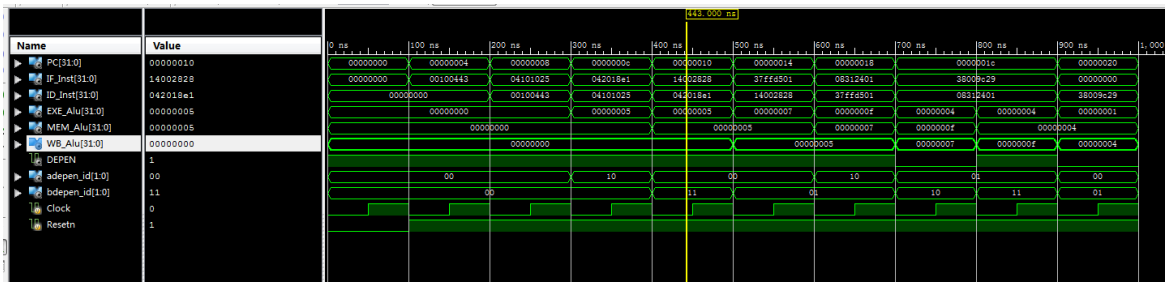
```
assign rom[6'h00]=32'h00000000;     //0地址为空，从1地址开始执行；
assign rom[6'h01]=32'h00100443;//add r1,r2,r3
assign rom[6'h02]=32'h04101025;//and r4,r1,r5
assign rom[6'h03]=32'h042018e1;//or r6,r7,r1
assign rom[6'h04]=32'h14002828;//addi r8,r1,0x000a
assign rom[6'h05]=32'h37ffd501;//load r1,0xfff5(r8)
assign rom[6'h06]=32'h08312401;//sll r9,r1,0x02
assign rom[6'h07]=32'h38009c29;//store r9,0x0027(r1)
```

8. 初始化寄存器和存储器

```
module Regfile(rna,rnb,d,wn,we,clk,clrn,qa,qb
    );
    input [4:0] rna,rnb,wn;
    input [31:0] d;
    input we,clk,clrn;
    output [31:0] qa,qb;
    reg [31:0] register [1:31];
    assign qa=(rna==0)?0:register[rna];
    assign qb=(rnb==0)?0:register[rnb];
    always @(posedge clk or negedge clrn)
        if(clrn==0)      //如果复位信号有效，则进行
        begin:init
        integer i;
        for(i=1;i<32;i=i+1)
            register[i]<=0;
        //初始化寄存器
        register[5'h01]<=32'h00000001;
        register[5'h02]<=32'h00000002;
        register[5'h03]<=32'h00000003;
        register[5'h04]<=32'h00000004;
        register[5'h05]<=32'h00000005;
        register[5'h06]<=32'h00000006;
        register[5'h07]<=32'h00000007;
        register[5'h08]<=32'h00000008;
        end
        else if((wn!=0)&&we)
            register[wn]<=d;
endmodule
```

```
module memory(we,addr,datain,clk,dataout
    );
    input [31:0] datain;
    input [4:0] addr;
    input clk,we;
    output [31:0] dataout;
    reg [31:0] ram [0:31];
    assign dataout=ram[addr];     //读出常有效
    always @(posedge clk)begin
    if(we)ram[addr]=datain;
    end

    integer i;
    initial begin     //存储器初始化
    for(i=0;i<32;i=i+1)
       ram[i]=0;
    ram[5'h01]=32'h00000001;
    ram[5'h02]=32'h00000002;
    ram[5'h03]=32'h00000003;
    ram[5'h04]=32'h00000004;
    ram[5'h05]=32'h00000005;
    ram[5'h06]=32'h00000006;
    ram[5'h07]=32'h00000007;
    ram[5'h08]=32'h00000008;

    end
endmodule
```

9. 仿真结果



由上图可知，在第 2 条指令进入流水线后，检测到 and 指令的 rs 与上一条 add 指令的 rd 冲突，故 adepen 信号变为 10，选择 2 号端口的 mem 级输入作为输入值参与运算；第 3 条指令进入流水线后，检测到 or 指令的 rt 与 and 指令的 rd 冲突，故 bdepen 信号变为 11，选择 3 号端口的 mem 级输出作为输入值参与运算。由此解决了数据冲突的问题。

十、 总结及心得体会：（联系理论知识进行说明）

通过本次实验，加深了对流水线处理器中存在的数据冒险的理解和掌握，学习了不同的软硬件的解决办法，提高了流水线处理机的性能。

十一、 对本实验过程及方法、手段的改进建议：

设计不同的、具有数据冒险的指令序列，观察指令流动状况，验证设计的正确性。

报告评分：

指导教师签字：

实　验　报　告

学生姓名：蒋芷昕　　学 号：2017180202005　　指导教师：王华

实验地点：主楼 A2-412　　实验时间：2020.9.20

一、 实验室名称：主楼 A2-412

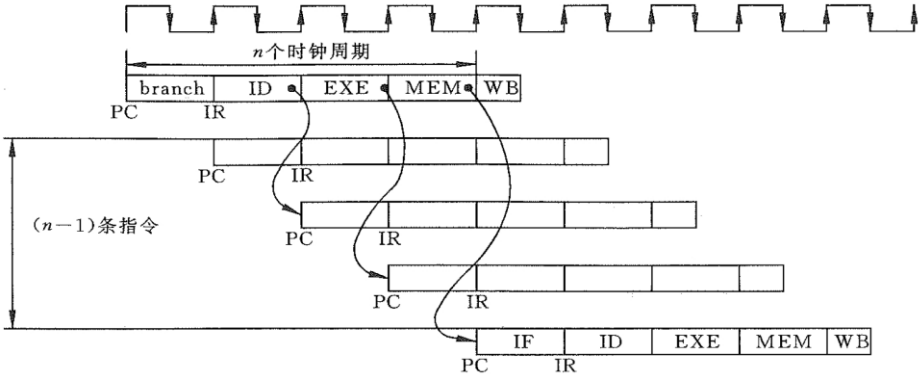
二、 实验项目名称：控制冒险问题

三、 实验学时：4 学时

四、 实验原理：（包括知识点，电路图，流程图）

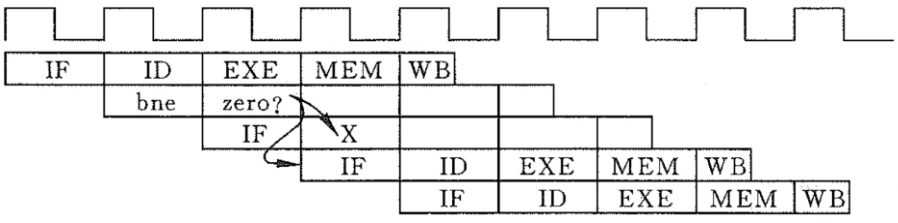
1. 控制冒险

由于流水线操作，在转移发生之前，若干条转移指令的后续指令已经被取到流水线处理机中。在进入下一个时钟周期取指令时，转移条件和转移 PC 不可用，这就是控制冒险。这样的后续指令的条数与转移指令执行完成需要多少个时钟周期有关。一般来说，若转移指令从取指令到执行完毕需要 n 个周期，则（n-1）条后续指令将受到影响。



2. 冻结或冲刷流水线

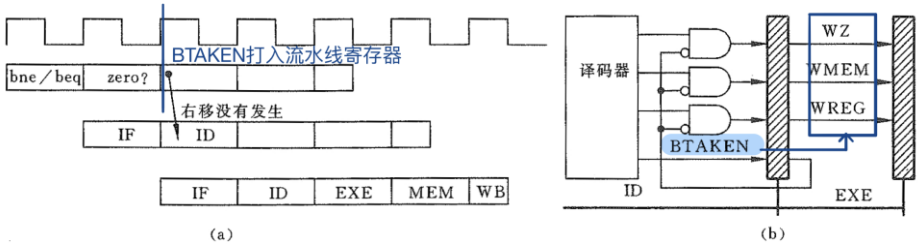
- 软件方法：即使在硬件方面不采取任何措施，编译器和汇编器只要在转移指令下面插入一条 nop 指令，就可以保证程序执行的正确性。
- 硬件方法：当发现当前指令是转移指令时，废弃当前取来的转移指令的后续一条指令，而并不封锁 PC 和 IR。



3. 假设转移不发生

针对条件转移指令，将该指令打入 IR 并让它执行下去，如果转移没有发

生，则让它继续执行；若发生转移，则终止执行。



4. 假设转移发生
   将所有分支看作选中分支，只要对分支指令进行译码并计算目标地址，就假定该分支将被选中，开始在目标位置提取和执行。
5. 延迟转移
   该方法总是执行转移指令的后续指令。首先在每一条转移指令下面安排一条 nop 指令，然后优化，即用一条有意义的、原来处于转移指令之前被执行的指令来替换增添的 nop 指令。若找不到这样的指令，则保留 nop。

五、 实验目的：

1. 进一步掌握流水线 CPU 和单周期 CPU 的区别；
2. 进一步熟悉 Verilog HDL 硬件设计语言；
3. 熟悉和掌握开发平台 Xilinx ISE Design Suite 14.7 集成开发系统的操作方法；
4. 进一步理解和掌握流水线控制冒险的概念和解决方法。

六、 实验内容：（介绍自己所选的实验内容）

1. 修改流水线 CPU 代码，解决无条件跳转指令（JUMP 指令）的控制冒险问题。消除无条件跳转指令的后续指令所产生的影响；
2. 修改流水线 CPU 代码，解决条件跳转指令（BNE 与 BEQ 指令）的控制冒险问题。
   a） 当条件跳转指令的 Z 信号还未准备好时，需要暂停流水线；
   b） 消除条件跳转指令的后续指令所产生的影响；
3. 对以下指令序列进行仿真，验证所实现流水线 CPU 能够解决控制冒险问题：

```
0x00:  nop;
0x04:  add  r1, r2, r3;
0x08:  and  r4, r1, r5;
0x0C:  or   r6, r7, r1;
0x10:  addi  r8, r1, 0x000a;
0x14:  load  r1, 0xfff5(r8);
0x18:  bne  r1, r8, 0x00000024;
0x1C:  sll  r9, r1, 0x02;
0x20:  store  r9, 0x0027(r1);
0x24:  jump  0x00000004;
```

4. 在流水线 CPU 结构图中做出相应修改：
   a) 画出为流水线解决数据冒险与控制冒险问题所增加的功能部件及相应控制信号；
   b) 说明所增加功能部件及相应控制信号是如何被使用。
5. 思考：检测所实现的流水线 CPU 代码是否能够正确运行以下指令序列（设 r1=1，r2=2），若不能，流水线应该如何扩展？

```
bne  r1, r1, xxxx;
beq  r2, r2, xxxx;
```

七、 实验器材（设备、元器件）：

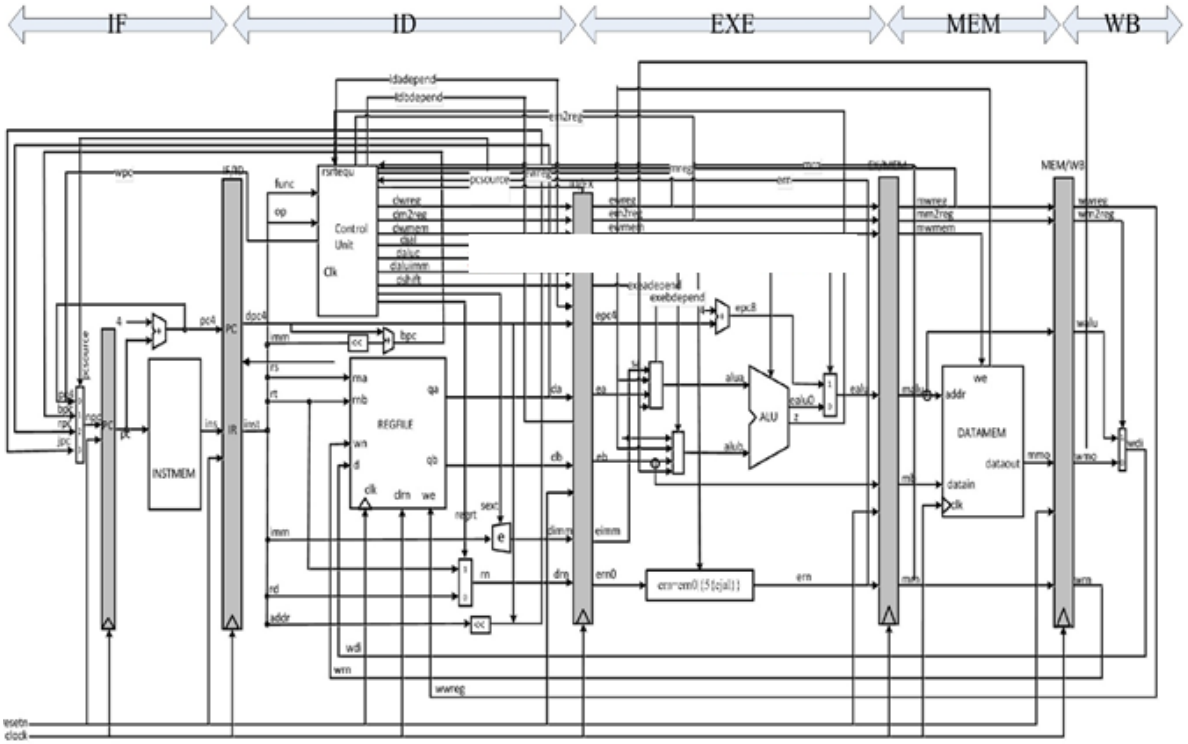
- 操作系统：Windows7（64 位）；
- 开发平台:Xilinx ISE Design Suite 14.7 集成开发系统；
- 下载软件：digilent.adept.system_v2.10.2.exe（由 FPGA 开用板厂家提供,用于将 Xilinx 开发生成的流代码 bit 文件下载到 FPGA 开发板上）；
- 编程语言：Verilog HDL 硬件描述语言。

八、 实验步骤：（编辑调试的过程）

1. 在实验三基础上补全控制冒险检测模块：废除直接跳转指令的下一条指令；根据 z 标志值判断废除条件跳转指令的下一条指令
2. 初始化寄存器和存储器
3. 分析流水线 CPU 中内部数据前推通路并仿真
4. 验证具有控制冒险的指令序列

九、 实验数据及结果分析：（实验运行结果介绍或者截图，对不同的结果进行分析）

1. 电路结构图



2. 在 PPCPU 顶层模块增加信号

```
,,
input Clock, Resetn;
output [31:0] PC, IF_Inst, ID_Inst;
output [31:0] EXE_Alu, MEM_Alu, WB_Alu;
output [1:0] exe_aop, exe_bop;
output stall, branch;


wire stall, id_lw, exe_lw,id_branch, branch;
```

3. 修改 ID 级

```
Control_Unit cu(rsrtequ,func,      //控制部件
           op,id_wreg,m2reg,wmem,aluc,regrt,aop,
           sext,pcsource,bop,
           exe_wreg,exe_rn,mem_wreg,mem_rn,wb_wreg,wb_rn,
           rs,rt,stall,zero,id_lw,exe_lw,branch,ir_branch);

Regfile rf (rs,rt,wdi,wb_rn,wb_wreg,clk,clrn,qa,qb);     //寄存器堆，有32个32位的寄存器，0号寄存器恒为0；在上升沿将数据写入寄存器
mux5_2_1 des_reg_num (rd,rt,regrt,id_rn); //       选择目的寄存器是来自于rd,还是rt，取决是寄存器类型指令或是Load指令；

//内部前推解决控制冒险可能的数据冒险，zero判断两书是否相等，然后传入Control_Unit
mux32_4_1 j_ina(qa, 0, exe_fw, mem_fw, aop, a);
mux32_4_1 j_inb(qb, 0, exe_fw, mem_fw, bop, b);
assign zero = (a^b == 32'h00000000);

assign e=sext&inst[25];//符号拓展或0拓展
assign ext16={16{e}};//符号拓展
assign imm={ext16,inst[25:10]};    //将立即数进行符号拓展

assign br_offset={imm[29:0],2'b00};      //计算偏移地址
add32 br_addr (pc4,br_offset,bpc);    //beq,bne指令的目标地址的计算
assign jpc={pc4[31:28],inst[25:0],2'b00};      //jump指令的目标地址的计算
```

4. 修改 CU 模块

```
//////////////////////////////////////////////控制信号的生成//////////////////////////////////////////////////////////////
assign branch = (i_beq & zero) | (i_bne & ~zero) | i_j;
assign wreg=~stall & (i_add|i_and|i_or|i_xor|i_sll|i_srl|i_addi|i_andi|i_ori|i_xori|i_lw) & ~ir_branch;    //寄存器写信号
assign wmem=i_sw & ~stall & ~ir_branch;//存储器写信号：为1时写存储器，否则不写

assign regrt=i_addi|i_andi|i_ori|i_xori|i_lw;    //regrt为1时目的寄存器是rt，否则为rd
assign m2reg=i_lw;  //运算结果写回寄存器：为1时将存储器数据写入寄存器，否则将ALU结果写入寄存器

assign sext=i_addi|i_lw|i_sw|i_beq|i_bne;//为1时符号拓展，否则零拓展

//判断当前rs、rt是否是源操作寄存器
assign id_rsIsReadReg = i_add|i_and|i_or|i_xor|i_addi|i_andi|i_ori|i_xori|i_lw|i_sw|i_beq|i_bne;
assign id_rtIsReadReg = i_add|i_and|i_or|i_xor|i_sll|i_srl|i_sw|i_beq|i_bne;

//数据前推ALU多路选择器控制信号
assign aop = {((exe_wreg & (exe_rn == rs)) | (mem_wreg & (mem_rn == rs))) & id_rsIsReadReg,
              i_sll | i_srl | (mem_wreg & (mem_rn == rs))};
assign bop = {((exe_wreg & (exe_rn == rt)) | (mem_wreg & (mem_rn == rt))) & id_rtIsReadReg,
              i_addi | i_andi | i_ori | i_xori | i_lw | i_sw | (mem_wreg & (mem_rn == rt))};

//load指令暂停控制信号，stall为暂停信号
assign exe_a_depen = (rs == exe_rn) && (exe_wreg == 1'b1) && (id_rsIsReadReg == 1'b1);
assign exe_b_depen = (rt == exe_rn) && (exe_wreg == 1'b1) && (id_rtIsReadReg == 1'b1);
assign stall = (exe_a_depen || exe_b_depen) & exe_lw;
```

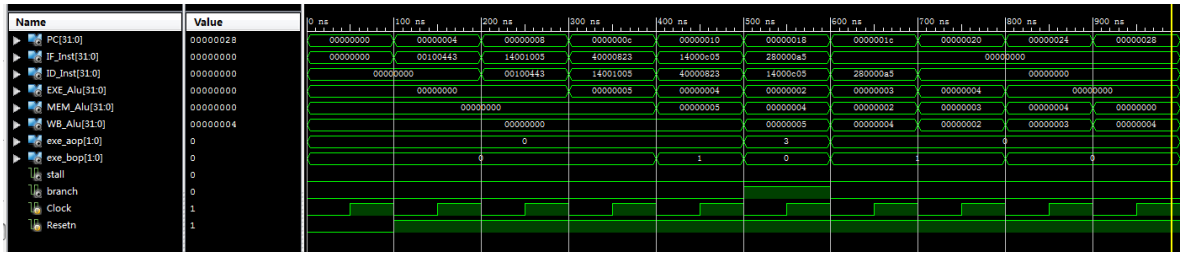
5. 修改 EXE 级模块

```
//////////////////////////////////////////////////////////////////////////////////
module execute(exe_aluc,exe_a,exe_b,exe_imm,id_aop,id_bop,exe_alu,z,  exe_fw, mem_fw
    );
    input [31:0] exe_a,exe_b,exe_imm, exe_fw, mem_fw;    //ea-由寄存器读出的操作数a；eb-由寄存器读出的操作数a；eimm-经过扩展的立即数；
    input [2:0] exe_aluc;    //ALU控制码
    input[1:0] id_aop,id_bop;    //ALU输入操作数的多路选择器
    output [31:0] exe_alu;    //alu操作输出
    output z;

    wire [31:0] alua,alub,sa;

    assign sa={27'b0,exe_imm[9:5]};//移位位数的生成

    //加上数据前推的exe级两操作数的四选一多路选择器
    mux32_4_1 alu_ina (exe_a, sa, exe_fw, mem_fw, id_aop, alua);
    mux32_4_1 alu_inb (exe_b, exe_imm, exe_fw, mem_fw, id_bop, alub);
    alu al_unit (alua,alub,exe_aluc,exe_alu,z);//ALU
```

6. 仿真测试

```
assign rom[6'h00]=32'h00000000;//nop
assign rom[6'h01]=32'h00100443;//add  r1,r2,r3;
assign rom[6'h02]=32'h14001005;//addi r5,r0,4;
assign rom[6'h03]=32'h40000823;//bne  6'06,r1,r3;
assign rom[6'h04]=32'h14000c05;//addi r5,r0,3;
assign rom[6'h05]=32'h14000861;//addi r1,r3,2;
assign rom[6'h06]=32'h280000a5;//ori  r5,r5,0;
assign rom[6'h07]=32'h00000000;
```

结果如下：



可以看到，bne 指令产生条件跳转之后，指令跳转到 6’h06 位置执行 ori 指令，其后面的 addi 指令虽然进入流水线执行 ，但并没有产生实际的影响，结果并没有改变。仿真结果符合预期。

十、 总结及心得体会：（联系理论知识进行说明）

通过本次实验，加深了对流水线处理器中存在的控制冒险的理解和掌握，学习了不同的软硬件的解决办法，提高了流水线处理机的性能。

对本实验过程及方法、手段的改进建议：

设计不同的、具有控制冒险的指令序列，观察指令流动状况，验证设计的正确性。

报告评分：

指导教师签字：