浙江大学

本科实验报告

课程名称：计算机组成

姓　　名：应周骏

学　　院：计算机科学与技术学院

系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3200103894

指导教师：马德

2022 年 6 月 20 日

浙江大学实验报告

课程名称：计算机组成 实验类型：综合

实验项目名称：CPU 设计之流水线

学生姓名：应周骏 专业：计算机科学与技术 学号：3200103894

同组学生姓名：无 指导老师：马德

实验地点：东 4-509 实验日期：2022 年 5 月 25 日

一、实验目的和要求

1.理解流水线 CPU 的基本原理和组织结构；

2.掌握五级流水线的工作过程和设计方法；

3.理解流水线 CPU 停机的原理；

4.设计流水线测试程序；

二、实验内容和原理

目标：

熟悉 RISC-V 五级流水线的工作特点，了解流水线处理器的原理，掌握 IP 核的使用方法，集成并测试 CPU。

内容：

集成设计流水线 CPU，在 Exp04 的基础上完成利用五级流水线各级封装模块集成 CPU 替换 Exp04 的单周期 CPU 为本实验集成的五级流水线 CPU，并设计流水线测试方案并完成测试。

原理：

1. 流水线优势

一个时钟周期完成一条指令所有操作，结构简单，但面对复杂指令集，其电路最长路径严重影响 CPU 工作频率，则效率太低。

一个时钟周期完成一条指令一个操作，面对单条指令会花费更多时间；但从

全局看，各个操作阶段的延时比整个 CPU 操作时钟延时短，时钟周期有效缩短。

在多周期基础上，利用不同阶段用不同时钟周期，功能部件可复用的特点，将不同指令的不同阶段重叠执行，则效率更高。

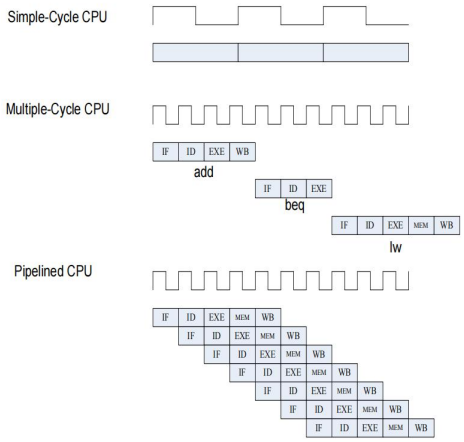

图 2.1 不同 CPU 策略

2. 流水线各阶段

取指：取指阶段涉及程序计数器（PC）和指令存储器（I_Mem）；程序计数器输出作为地址从指令存储器中读取指令；

IF_reg_ID：暂存指令和 PC 值，以待下一级使用；

译码：译码阶段涉及寄存器堆（RegisterFiles）和译码器、立即数生成单元（ImmGen）；从寄存器堆可以读取操作数，译码器对指令进行解析产生各种各种控制信号，立即数生成单元根据控制信号和输入指令生成各种类型的立即数。

ID_reg_Ex：暂存 PC 值，寄存器读取的数据，立即数和控制信号以待下一级使用；

执行：执行阶段涉及运算单元（ALU）它获取操作数并完成指定的算数运算或逻辑运算；

Ex_reg_Mem：暂存运算结果和控制信号，以待下一级使用；

存储器访问：存储器访问阶段涉及数 据 存 储 器（ D_Mem ）；Load\Store 指令对数据存储器进行读或写；

Mem_reg_WB：暂存存储器结果和控制信号，以待下一级使用；

写回：写回阶段涉及寄存器堆（RegisterFiles）；将 ALU 的运算结果、存储器输出结果、PC+4 写回到寄存器堆。

写回阶段结束，一次完整的五级流水操作完成；此时下一次操作进行到存储

器访问阶段（如果有）。由于在各级流水线之间插入了寄存器作为数据及控制信号的暂存，从而实现多条指令的重叠而不受影响。

3. 流水线控制

取指控制：取指阶段，读指令存储器和写 PC 值永远有效，无需控制信号。

译码控制：译码阶段，立即数生成单元需要根据指令类型产生对应输出，ImmSel 信号输出；其他信号暂存寄存器。

执行控制：执行阶段，ALU 的操作和第二个操作数 Src_B 需要选择，ALU_ctrl、ALUSrc_B 信号输出；其他信号暂存寄存器。

存储器访问控制：存储器访问阶段，需要读写存储器以及根据分支跳转指令判定选择 PC 转移值，MemRW、Branch、BranchN、Jump 输出；其他信号暂存。

写回控制：写回阶段， ALU 运算结果、存储器输出等需要选择写回寄存器堆，同时寄存器堆的写使能需要设置；MemToReg、RegWrite 信号输出。

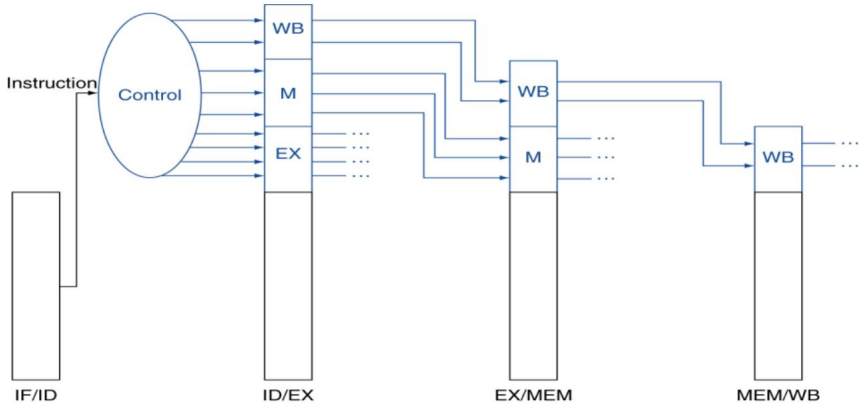

图 2.2 流水线控制示意图

三、实验过程和数据记录

1. 工程文件建立

新建工程文件，命名为“OxExp05_pipeline”。

2. CSSTE 模块更新

在实验二的“CSSTE. v”文件基础上，修改部分代码，主要集中在 VGA 和 pipeline_CPU 接口（详见附件）。

```
VGA U11
    (.Addr_out(Addr_out),
     .clk_100m(clk_100mhz),
     .clk_25m(clkdiv[1]),
     .Data_out(Data_out),
     .Data_out_WB(Data_out_WB),
     .hs(HSYNC),
     .MemRW_Mem(MemRW_Mem),
     .MemRW_Ex(MemRW_EX),
     .PC_IF(PC_out_IF),
     .PC_ID(PC_out_ID),
     .PC_Ex(PC_out_EX),
     .inst_ID(Inst_ID),
     .inst_IF(Inst_IF),
     .rst(rst),
     .vga_b(Blue),
     .vga_g(Green),
     .vga_r(Red),
     .vs(VSYNC));
```

图 3.2.1 VGA 更新

```
Pipeline_CPU U1
      (.Addr_out(Addr_out),
       .Data_in(Data_in),
       .Data_out(Data_out),
       .Data_out_WB(Data_out_WB),
       .MIO_ready(1'b0),
       .MemRW_Mem(MemRW_Mem),
       .MemRW_EX(MemRW_EX),
       .PC_out_IF(PC_out_IF),
       .PC_out_ID(PC_out_ID),
       .PC_out_EX(PC_out_EX),
       .inst_ID(Inst_ID),
       .clk(Clk_CPU),
       .inst_IF(Inst_IF),
       .rst(rst));
```

图 3.2.2 Pipeline_CPU 接口

3. Pipeline_CPU 模块设计

新建“Pipeline_CPU.v”，依据给出的原理图，连接各给出模块。

```
module Pipeline_CPU(
    input[31:0] Data_in,
    input rst,
    input clk,
    input[31:0] inst_IF,
    input MIO_ready,

    output[31:0] PC_out_EX,
    output[31:0] PC_out_ID,
    output[31:0] PC_out_IF,
    output[31:0] inst_ID,
    output[31:0] Addr_out,
    output[31:0] Data_out,
    output[31:0] Data_out_WB,
    output MemRW_Mem,
    output MemRW_EX,
    output CPU_MIO
    );

wire[31:0]Inst_in_ID,  PC_out_IFID,inst_out_IFID;
wire PCSrc,  RegWrite_out_MemWB;

wire[31:0]Rs1_out_ID,Rs2_out_ID,Imm_out_ID;
wire[4:0]Rd_addr_out_ID;
wire [3:0]ALU_control_ID;
wire[1:0]MemtoReg_ID,Jump_ID;
wire ALUSrc_B_ID,Branch_ID,BranchN_ID,MemRW_ID,RegWrite_out_ID;

wire[31:0]PC_out_IDEX,Rs1_out_IDEX,Rs2_out_IDEX,Imm_out_IDEX;
wire[4:0]Rd_addr_out_IDEX;
wire[1:0]MemtoReg_out_IDEX,Jump_out_IDEX;
wire[3:0]ALU_control_out_IDEX;
wire ALUSrc_B_out_IDEX,Branch_out_IDEX,BranchN_out_IDEX,MemRW_out_IDEX,RegWrite_out_IDEX;

wire[31:0]PC4_out_EX,ALU_out_EX,Rs2_out_EX;
wire zero_out_EX;

wire[31:0]PC_out_EXMem,PC4_out_EXMem,ALU_out_EXMem;
wire[4:0]Rd_addr_out_EXMem;
wire[1:0]MemtoReg_out_EXMem,Jump_out_EXMem;
wire zero_out_EXMem,Branch_out_EXMem,BranchN_out_EXMem,MemRW_out_EXMem,RegWrite_out_EXMem;

wire[31:0]PC4_out_MemWB,ALU_out_MemWB,DMem_data_out_MemWB;
wire[4:0]Rd_addr_out_MemWB;
wire[1:0]MemtoReg_out_MemWB;
```

图 3.3.1 流水线 CPU 接口

```
Pipeline_IF Instruction_Fetch(
    .clk_IF(clk),
    .rst_IF(rst),
    .en_IF(1'b1),
    .PC_in_IF(PC_out_EXMem),
    .PCSrc(PCSrc),
    .PC_out_IF(PC_out_IF)
);

IF_reg_ID IF_reg_ID(
    .clk_IFID(clk),
    .rst_IFID(rst),
    .en_IFID(1'b1),
    .PC_in_IFID(PC_out_IF),
    .inst_in_IFID(inst_IF),
    .PC_out_IFID(PC_out_ID),
    .inst_out_IFID(Inst_in_ID)
);

Pipeline_ID Instruction_Decoder(
    .clk_ID(clk),
    .rst_ID(rst),
    .RegWrite_in_ID(RegWrite_out_MemWB),
    .Rd_addr_ID(Rd_addr_out_MemWB),
    .Wt_data_ID(Data_out_WB),
    .Inst_in_ID(Inst_in_ID),
    .Rd_addr_out_ID(Rd_addr_out_ID),
    .Rs1_out_ID(Rs1_out_ID),
```

图 3.3.2 连接各模块（详细代码见附件）

4. 五级流水线模块及寄存器设计

新建“Pipeline_IF.v”模块，实现取指模块。

```
module Pipeline_IF(
    input clk_IF,
    input rst_IF,
    input en_IF,
    input [31:0]PC_in_IF,
    input PCSrc,
    output [31:0]PC_out_IF
    );

    wire[31:0] U0_out, U2_out;
    MUX2T1_32 U0(
        .I0(U2_out),
        .I1(PC_in_IF),
        .s(PCSrc),
        .O(U0_out)
    );

    REG32 U1(
        .clk(clk_IF),
        .rst(rst_IF),
        .CE(en_IF),
        .D(U0_out),
        .Q(PC_out_IF)
    );

    add_32 U2(
        .a(32'h4),
        .b(PC_out_IF),
        .c(U2_out)
    );
```

图 3.4.1 取指模块代码

新建“IF_reg_ID.v”模块，实现取指/译码寄存器模块。

```
module IF_reg_ID(
    input clk_IFID,
    input rst_IFID,
    input en_IFID,
    input [31:0]PC_in_IFID,
    input [31:0]inst_in_IFID,
    output reg [31:0]PC_out_IFID,
    output reg [31:0]inst_out_IFID
    );
    always @(posedge clk_IFID or posedge rst_IFID)
    if (rst_IFID==1'b1) begin
        PC_out_IFID <= 32'h0;
        inst_out_IFID <= 32'h0;
    end
    else if (en_IFID==1'b1) begin
        PC_out_IFID <= PC_in_IFID;
        inst_out_IFID <= inst_in_IFID;
    end

endmodule
```

图 3.4.2 IF/ID.reg 实现

新建“Pipeline_ID.v”模块，实现译码模块，并修改相应的译码模块，支持该实验需要，但是扩展指令支持，而基础指令不支持的功能。

```
wire[2:0] ImmSel;
assign Rd_addr_out_ID = Inst_in_ID[11:7];

regs Regs_0(
    .clk(clk_ID),
    .rst(rst_ID),
    .Rs1_addr(Inst_in_ID[19:15]),
    .Rs2_addr(Inst_in_ID[24:20]),
    .Wt_addr(Rd_addr_ID),
    .Wt_data(Wt_data_ID),
    .RegWrite(RegWrite_in_ID),
    .Rs1_data(Rs1_out_ID),
    .Rs2_data(Rs2_out_ID)
);

ImmGen ImmGen_0(
    .ImmSel(ImmSel),
    .inst_field(Inst_in_ID),
    .Imm_out(Imm_out_ID)
);
```

```
SCPU_ctrl_more SCPU_ctrl_0(
    .OPcode(Inst_in_ID[6:2]),
    .Fun3(Inst_in_ID[14:12]),
    .Fun7(Inst_in_ID[30]),
    .ImmSel(ImmSel),
    .ALUSrc_B(ALUSrc_B_ID),
    .MemtoReg(MemtoReg_ID),
    .Jump(Jump_ID),
    .Branch(Branch_ID),
    .BranchN(BranchN_ID),
    .RegWrite(RegWrite_out_ID),
    .MemRW(MemRW_ID),
    .ALU_Control(ALU_control_ID)
```

图 3.4.3 译码模块代码

新建“ID_reg_EX.v”模块，实现译码/执行寄存器模块。

```
always @(posedge clk_IDEX or posedge rst_IDEX) begin
    if(rst_IDEX == 1'b1)begin
        PC_out_IDEX <= 32'h0;
        Rd_addr_out_IDEX <= 5'h0;
        Rs1_out_IDEX <= 32'h0;
        Rs2_out_IDEX <= 32'h0;
        Imm_out_IDEX <= 32'h0;
        ALUSrc_B_out_IDEX <= 1'h0;
        ALU_control_out_IDEX <= 4'h0;
        Branch_out_IDEX <= 1'h0;
        BranchN_out_IDEX <= 1'h0;
        MemRW_out_IDEX <= 1'h0;
        Jump_out_IDEX <= 2'h0;
        MemtoReg_out_IDEX <= 2'h0;
        RegWrite_out_IDEX <= 1'h0;
    end
    else if(en_IDEX == 1'b1) begin
        PC_out_IDEX <= PC_in_IDEX;
        Rd_addr_out_IDEX <= Rd_addr_IDEX;
        Rs1_out_IDEX <= Rs1_in_IDEX;
        Rs2_out_IDEX <= Rs2_in_IDEX;
        Imm_out_IDEX <= Imm_in_IDEX;
        ALUSrc_B_out_IDEX <= ALUSrc_B_in_IDEX;
        ALU_control_out_IDEX <= ALU_control_in_IDEX;
        Branch_out_IDEX <= Branch_in_IDEX;
        BranchN_out_IDEX <= BranchN_in_IDEX;
        MemRW_out_IDEX <= MemRW_in_IDEX;
        Jump_out_IDEX <= Jump_in_IDEX;
        MemtoReg_out_IDEX <= MemtoReg_in_IDEX;
        RegWrite_out_IDEX <= RegWrite_in_IDEX;
    end
```

图 3.4.4 ID/EX.reg 实现

新建“Pipeline_EX.v”模块，实现执行模块。

```
wire[31:0] mux_out;
assign Rs2_out_EX = Rs2_in_EX;
add_32 add_32_0(
    .a(PC_in_EX),
    .b(Imm_in_EX),
    .c(PC_out_EX)
);

add_32 add_32_1(
    .a(32'h4),
    .b(PC_in_EX),
    .c(PC4_out_EX)
);

ALU ALU(
    .A(Rs1_in_EX),
    .ALU_operation(ALU_control_in_EX),
    .B(mux_out),
    .res(ALU_out_EX),
    .zero(zero_out_EX)
);

MUX2T1_32 MUX2T1_32_0(
    .I0(Rs2_in_EX),
    .I1(Imm_in_EX),
    .s(ALUSrc_B_in_EX),
    .O(mux_out)
);
```

图 3.4.5 执行模块代码

新建“EX_reg_MEM.v”模块，实现执行/存储器访问寄存器模块。

```
always @(posedge clk_EXMem or posedge rst_EXMem) begin
    if(rst_EXMem == 1'b1) begin
        PC_out_EXMem <= 32'h0;
        PC4_out_EXMem <= 32'h0;
        Rd_addr_out_EXMem <= 5'h0;
        zero_out_EXMem <= 1'h0;
        ALU_out_EXMem <= 32'h0;
        Rs2_out_EXMem <= 32'h0;
        Branch_out_EXMem <= 1'h0;
        BranchN_out_EXMem <= 1'h0;
        MemRW_out_EXMem <= 1'h0;
        Jump_out_EXMem <= 2'h0;
        MemtoReg_out_EXMem <= 2'h0;
        RegWrite_out_EXMem <= 1'h0;
    end
    else if(en_EXMem == 1'b1) begin
        PC_out_EXMem <= PC_in_EXMem;
        PC4_out_EXMem <= PC4_in_EXMem;
        Rd_addr_out_EXMem <= Rd_addr_EXMem;
        zero_out_EXMem <= zero_in_EXMem;
        ALU_out_EXMem <= ALU_in_EXMem;
        Rs2_out_EXMem <= Rs2_in_EXMem;
        Branch_out_EXMem <= Branch_in_EXMem;
        BranchN_out_EXMem <= BranchN_in_EXMem;
        MemRW_out_EXMem <= MemRW_in_EXMem;
        Jump_out_EXMem <= Jump_in_EXMem;
        MemtoReg_out_EXMem <= MemtoReg_in_EXMem;
        RegWrite_out_EXMem <= RegWrite_in_EXMem;
    end
end
```

图 3.4.6 EX/MEM.reg 实现

新建“Pipeline_MEM.v”模块，实现存储器访问模块。

```
module Pipeline_Mem(
    input zero_in_Mem,
    input Branch_in_Mem,
    input BranchN_in_Mem,
    input [1:0]Jump_in_Mem,
    output PCSrc
    );

    assign PCSrc = (((~zero_in_Mem) & BranchN_in_Mem) | ((zero_in_Mem) & Branch_in_Mem)) | Jump_in_Mem[0];

endmodule
```

图 3.4.7 存储器访问模块代码

新建“MEM_reg_WB.v”模块，实现存储器访问/写回寄存器模块。

```
    output reg[31:0] PC4_out_MemWB, //PC+4输出
    output reg[4:0] Rd_addr_out_MemWB, //写目的地址输出
    output reg[31:0] ALU_out_MemWB, //ALU输出
    output reg[31:0] DMem_data_out_MemWB, //存储器数据输出
    output reg[1:0] MemtoReg_out_MemWB, //写回
    output reg RegWrite_out_MemWB //寄存器堆读写);
    );

    always @(posedge clk_MemWB or posedge  rst_MemWB) begin
        if(rst_MemWB == 1'b1) begin
            PC4_out_MemWB <= 32'h0;
            Rd_addr_out_MemWB <= 5'h0;
            ALU_out_MemWB <= 32'h0;
            DMem_data_out_MemWB <= 32'h0;
            MemtoReg_out_MemWB <= 2'h0;
            RegWrite_out_MemWB <= 1'h0;
        end
        else if(en_MemWB == 1'b1) begin
            PC4_out_MemWB <= PC4_in_MemWB;
            Rd_addr_out_MemWB <= Rd_addr_MemWB;
            ALU_out_MemWB <= ALU_in_MemWB;
            DMem_data_out_MemWB <= DMem_data_MemWB;
            MemtoReg_out_MemWB <= MemtoReg_in_MemWB;
            RegWrite_out_MemWB <= RegWrite_in_MemWB;
        end
    end
endmodule
```

图 3.4.8 MEM/WB.reg 文件

新建“Pipeline_WB.v”模块，实现写回模块。

```
module Pipeline_WB(
    input[31:0] PC4_in_WB, //PC+4输入
    input[31:0] ALU_in_WB, //ALU结果输出
    input[31:0] DMem_data_WB, //存储器数据输入
    input[1:0] MemtoReg_in_WB, //写回选择控制
    output [31:0] Data_out_WB //写回数据输出
    );

    MUX4T1_32 MUX4T1_32_0(
        .s(MemtoReg_in_WB),
        .I0(ALU_in_WB),
        .I1(DMem_data_WB),
        .I2(PC4_in_WB),
        .I3(PC4_in_WB),
        .O(Data_out_WB)
    );

endmodule
```

图 3.4.9 写回模块代码

5. 模块集成

在工程中加入实验二中相关的模块，以及新的 VGA 模块，修改 VgaDisplay 模块的相关文件地址为本机实际存储地址。同时，依照此前实验，生成流水线测试指令/冒险测试指令的 ROM 文件，本实验将不含冒险的指令作为测试。

```
(* ram_style = "block" *) reg [7:0] display_data[0:4095];
initial $readmemh("D://14.7//LabofCOAD//project5//OExp02-IP2SOC_pipline//vga_debugger.mem", display_data);

wire [11:0] text_index = (vga_y / 16) * 80 + vga_x / 8;
// I don't know why I need this '- (vga_y / 16)' ...
wire [7:0] text_ascii = display_data[text_index] - (vga_y / 16);
wire [2:0] font_x = vga_x % 8;
wire [3:0] font_y = vga_y % 16;
wire [11:0] font_addr = text_ascii * 16 + font_y;

(* ram_style = "block" *) reg [7:0] fonts_data[0:4095];
initial $readmemh("D://14.7//LabofCOAD//project5//OExp02-IP2SOC_pipline//font_8x16.mem", fonts_data);
wire [7:0] font_data = fonts_data[font_addr];
```

图 3.5.1 VGA 模块更新

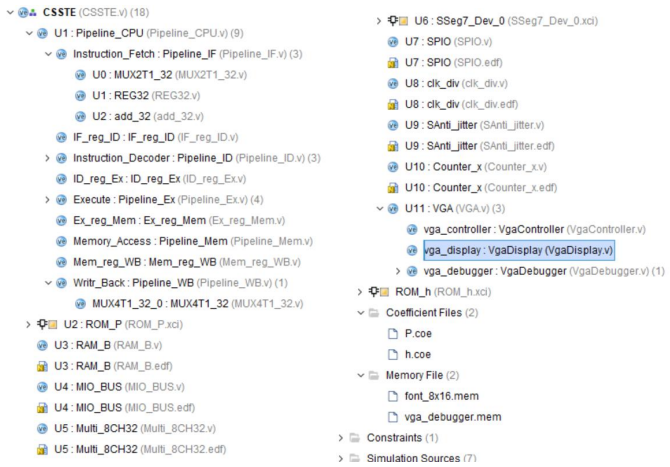

图 3.5.2 整体模块架构

6. 物理验证

对上述流水线 CPU 生成 bit 文件，在 SWORD 实验板上进行验证。

四、实验结果分析

1. 各模块设计思路

对于五级流水线的具体模块，按照 PPT 给出的原理图进行设计。其中，由于 PPT 给出的整体接口位宽与拓展指令/基础指令均不兼容，故本实验采取了扩展指令模块，并对接口做了相应调整，以实现 BranchN 指令。

对于寄存器，本实验采用时序逻辑对相应控制信号和数值进行保持/清零。

2. 物理验证结果

物理验证结果能够通过验收。同时，由于本实验设计的控制通路模块，对无需跳转的 PC 值的 ALU 结果仍然输出有效值，及有值但不用，所以输出 PC 时在最后一级的结果存在不符合 PC+4 的条件，属于合理情况。

五、讨论与心得

1. 通过本次实验，我掌握了流水线 CPU 的基本运作原理，基本了解了五级流水线 CPU 的工作流程。

2. 本次实验中，由于模块之间信号位宽与此前的扩展/基础实验均不一致，因此在调整位宽时花了较多时间，不过整体流程相对顺利，没有遇到很多问题。

3. 本次实验由于时间限制没有进一步实现冒险，希望在暑假有空时进行完善和学习，为未来进一步的硬件学习打下基础。