浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：汪辉

学　　院：计算机科学与技术学院

系：计算机系

专　　业：计算机科学与技术

学　　号：3190105609

指导教师：陈文智

2021 年 12 月 21 日

浙江大学实验报告

课程名称：计算机体系结构 实验类型：综合

实验项目名称：Pipelined CPU supporting multi-cycle operations

学生姓名：王嘉豪 专业：计算机科学与技术 学号：3190105304

同组学生姓名：汪辉 指导老师：陈文智

实验地点：曹西 301 实验日期：2021 年 12 月 21 日

一、 实验目的和要求

- 了解支持多周期计算的流水线原理
- 掌握支持多周期计算的流水线的设计方法
- 掌握支持多周期计算的流水线的验证方法

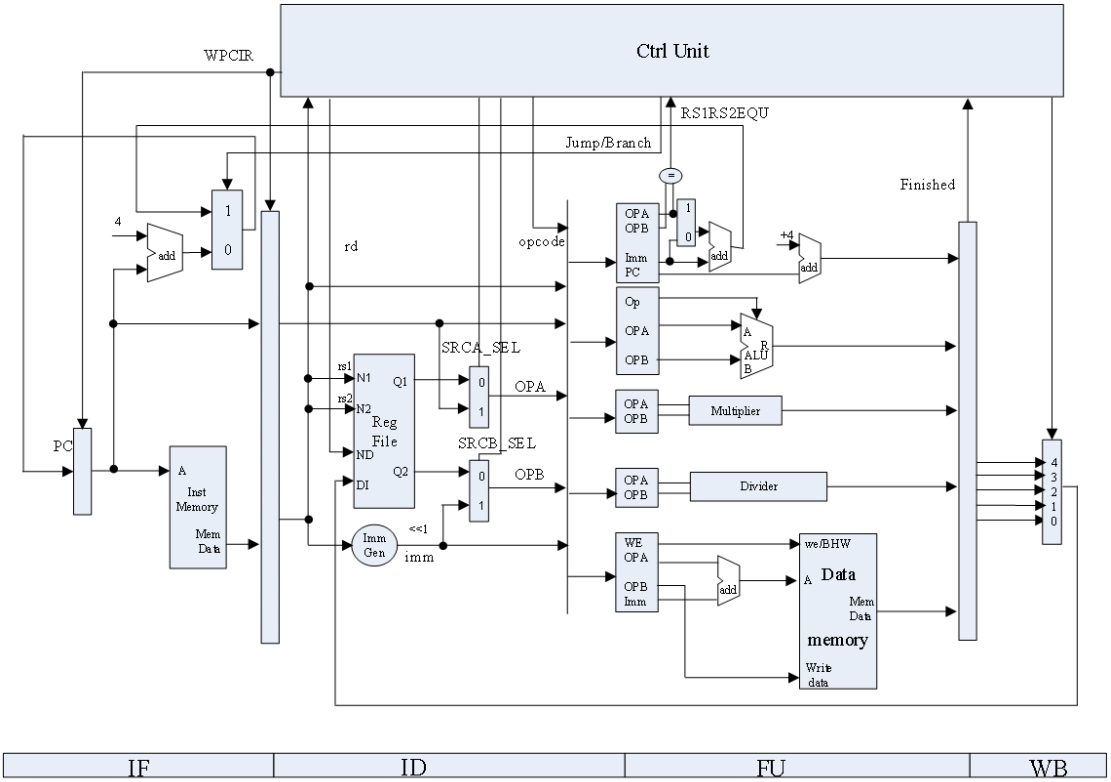
二、 实验内容和原理

1. 实验内容：

- 利用 IF/ID/FU/WB 模块和 FU 模块重新设计支持多周期计算的流水线
- 重新设计 CPU 的控制模块
- 通过程序验证流水线 CPU 并且观察程序的计算过程

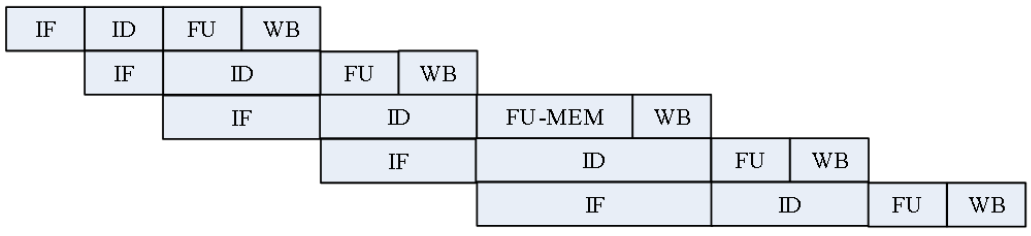
2. 实验原理：

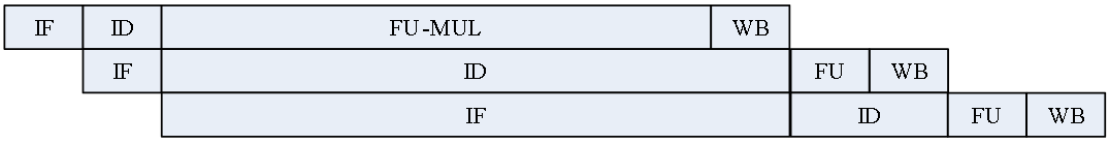
2.1 Pipelined CPU



将不同的计算方式分开分别设计模块，当需要计算时，将需求发送给对应的模块。模块被占用期间，相同类型计算指令可以被阻塞。最后的结果通过多路选择器一个个输出。

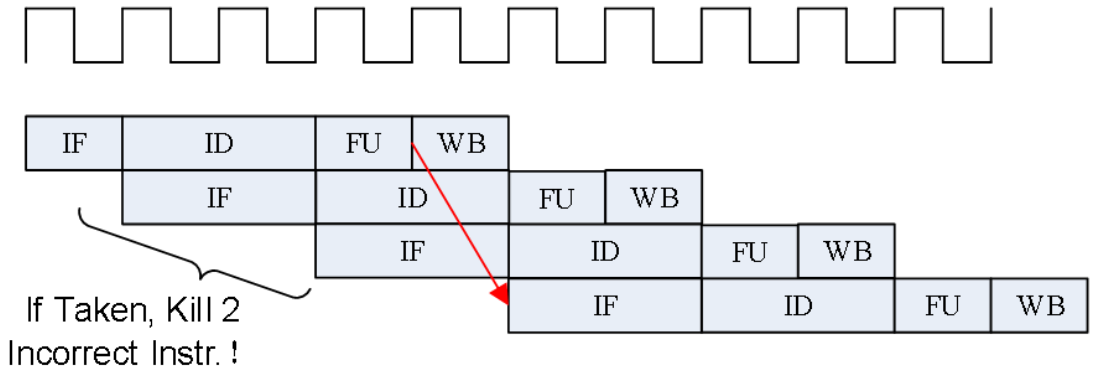
2.2 Pipelines resolving Data Hazards





遇到数据竞争时，插入 stall 等到竞争不再存在，再继续运行。

4.3 Methods of resolving Control hazards



是否跳转和地址的计算在 FU 里面进行，采取预测不跳转的方式，假如最后需要跳转，那么就需要清理掉两条已经读入的错误指令。

三、　实验过程和数据记录

本实验主要是需要补充 FU_ALU.v，FU_div.v，FU_jump.v，FU_mem.v，FU_mul.v 以及 RV32core.v，根据相关文件功能，依次进行补充即可。

1. FU_ALU

当 state=0 时，计算还未完成，分别将两个操作数 ALUA、ALUB 以及操作符 ALUControl 的值赋给 A，B 以及 Control，并将 state 置 1，标志下个周期完成计算。

```
always@(posedge clk) begin
    if(EN & ~state) begin // state == 0
        Control <= ALUControl ;
        A <= ALUA ;
        B <= ALUB ;
        //to fill sth.in
        state <= 1;
    end
    else state <= 0;
end
```

2. FU_mul

乘法器的补充部分需要控制乘法执行七个周期，根据给出的 state[6:0]寄存器，补充代码如下。每过一个周期，状态寄存器发生改变。这里直接使用移位操作符，避免较多的选择分支。

```
wire[31:0] mulres;
always@(posedge clk) begin
```

```
        if(EN & state == 0) begin // state == 0
            A_reg <= A;
            B_reg <= B;
            state[6] <= 1;
        end
        else state <= (state >> 1);
    end
```

3. FU_div

当除法器被调用且处于未完成状态时，持续给除法模块输入除数被除数以及将它们的 valid 置为 1，当计算完成时，res_valid 为 1，此时将除数被除数的 valid 置为 0。

```
    always@(posedge clk) begin
        if(EN & ~state & ~res_valid) begin // state == 0
            A_reg <= A;
            A_valid <= 1;
            B_reg <= B;
            B_valid <= 1;
            state <= 1;
        end
        else if(res_valid) begin
            A_valid <= 0;
            B_valid <= 0;
            state <= 0;
        end
    end
```

4. FU_mem

在第一个周期准备好内存操作需要的参数，经过两个周期进行内存访问以及数据的存取。

```
    reg[31:0] addr ;
    always@(posedge clk) begin
        if( EN && state==0 ) begin // state == 0
            mem_w_reg <= mem_w ;
            bhw_reg <= bhw ;
            rs1_data_reg <= rs1_data ;
            rs2_data_reg <= rs2_data ;
            imm_reg <= imm ;
            addr <= rs1_data+imm ;
            state <= 2;
        end
        else if ( state == 2 ) begin
            state <= 1;
```

```
        end
        else if ( state == 1 ) state <= 0 ;
    end
```

5. FU_jump

通过给 cmp 模块传入比较的值和比较方式，得出是否需要进行跳转。根据跳转指令类型，mux 模块选择下一周期的 PC 值。

```
    reg[31:0] jalr_pc, b_jal_pc ;

    wire JAL = cmp_ctrl_reg == 3'b000 ;
    wire BEQ = cmp_ctrl_reg == 3'b001 ;
    wire BNE = cmp_ctrl_reg == 3'b010 ;
    wire BLT = cmp_ctrl_reg == 3'b011 ;
    wire BGE = cmp_ctrl_reg == 3'b100 ;
    wire BLTU = cmp_ctrl_reg == 3'b101 ;
    wire BGEU = cmp_ctrl_reg == 3'b110 ;

    always@(posedge clk) begin
        if(EN & ~state) begin // state == 0
            state <= 1;
            JALR_reg <= JALR ;
            cmp_ctrl_reg <= cmp_ctrl ;
            rs1_data_reg <= rs1_data ;
            rs2_data_reg <= rs2_data ;
            if ( JALR ) PC_jump_reg <= rs1_data_reg+imm ;
            else PC_jump_reg <= PC+imm ;
            PC_wb_reg <= PC+4 ;
        end
        else state <= 0;
    end

    wire EQUAL = rs1_data_reg == rs2_data_reg ;
    wire NEQUAL = ~( rs1_data_reg == rs2_data_reg ) ;
    wire LTU = rs1_data_reg < rs2_data_reg ;
    wire GEU = rs1_data_reg >= rs2_data_reg ;
    wire LT =  rs1_data_reg[31]&&~rs2_data_reg[31] ||

rs1_data_reg[31]&&rs2_data_reg[31]&&rs1_data_reg>rs2_data_
reg ||

~rs1_data_reg[31]&&~rs2_data_reg[31]&&rs1_data_reg<rs2_dat
a_reg ;
    wire GE =  ~( rs1_data_reg[31]&&~rs2_data_reg[31] ||
```

```
rs1_data_reg[31]&&rs2_data_reg[31]&&rs1_data_reg>rs2_data_
reg ||

~rs1_data_reg[31]&&~rs2_data_reg[31]&&rs1_data_reg<rs2_dat
a_reg ) ;

   assign cmp_res = JAL | JALR_reg |
                BEQ & EQUAL |
                BNE & NEQUAL |
                BLT & LT |
                BGE & GE |
                BLTU & LTU |
                BGEU & GEU ;

   assign PC_jump = PC_jump_reg ;
   assign PC_wb = PC_wb_reg ;
```

6. RV32core

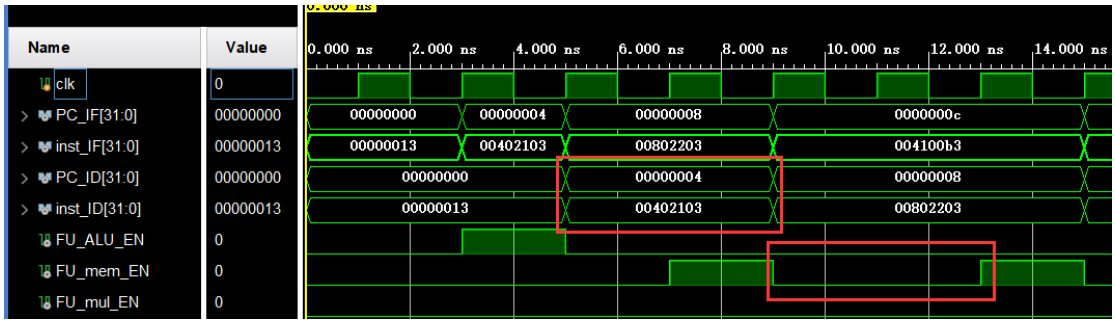
如下补全即可。

```
ImmGen
imm_gen(.inst_field(inst_ID),.ImmSel(ImmSel_ctrl),.Imm_out
(Imm_out_ID));

MUX2T1_32
mux_imm_ALU_ID_A(.I0(rs1_data_ID),.I1(Imm_out_ID),.s(ALUSr
cA_ctrl),.o(ALUA_ID));

MUX2T1_32
mux_imm_ALU_ID_B(.I0(rs2_data_ID),.I1(Imm_out_ID),.s(ALUSr
cB_ctrl),.o(ALUB_ID));

MUX8T1_32
mux_DtR(.s(DatatoReg_ctrl),.I1(ALUout_WB),.I2(mem_data_WB)
,.I3(mulres_WB),.I4(divres_WB),.I5(PC_wb_WB),.o(wt_data_WB
));
```

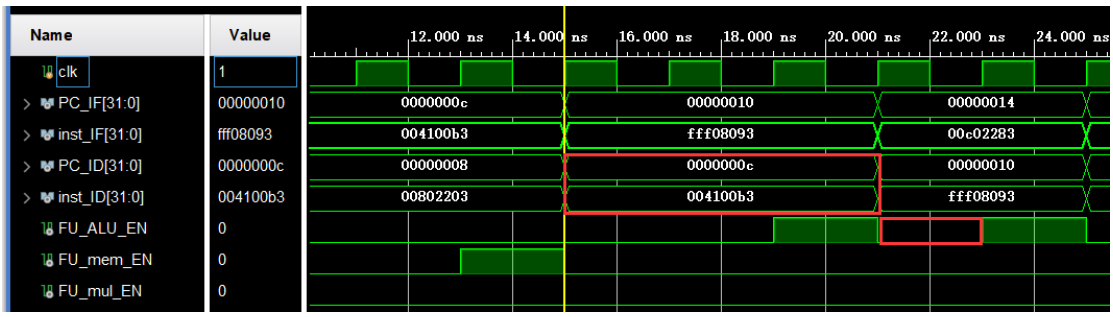
四、　实验结果分析

1. 仿真结果：

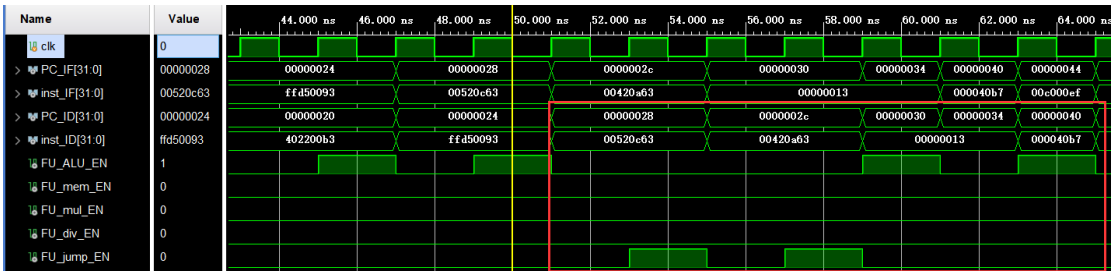
FU_mem：当 ID 阶段译码为数据的存取时，FU_mem_EN 被置为 1，之后使用两个周期进行相关操作，再进行下一条指令的执行。



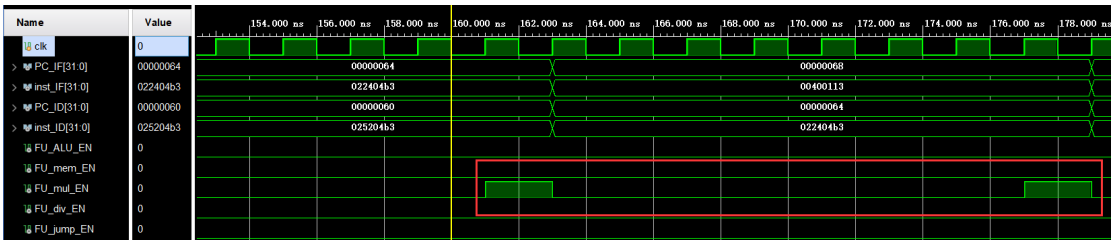
FU_ALU：当 ID 阶段译码为 ALU 进行数据运算时，FU_ALU_EN 被置为 1，之后使用一个周期进行相关操作，再进行下一条指令的执行。



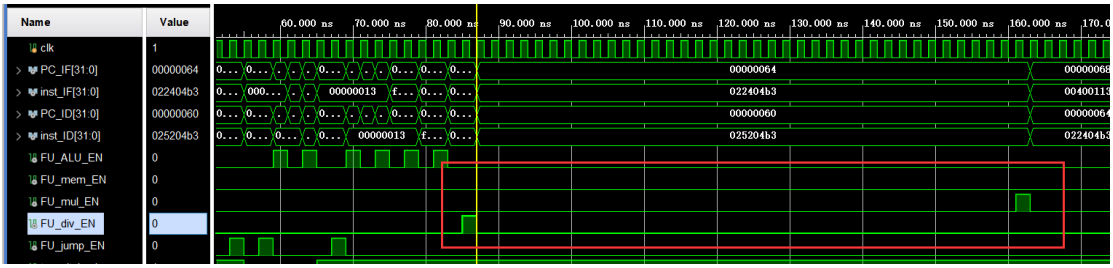
FU_jump：当 ID 阶段译码为跳转相关时，FU_jump_EN 被置为 1，之后使用一个周期进行相关操作，再进行下一条指令的执行。



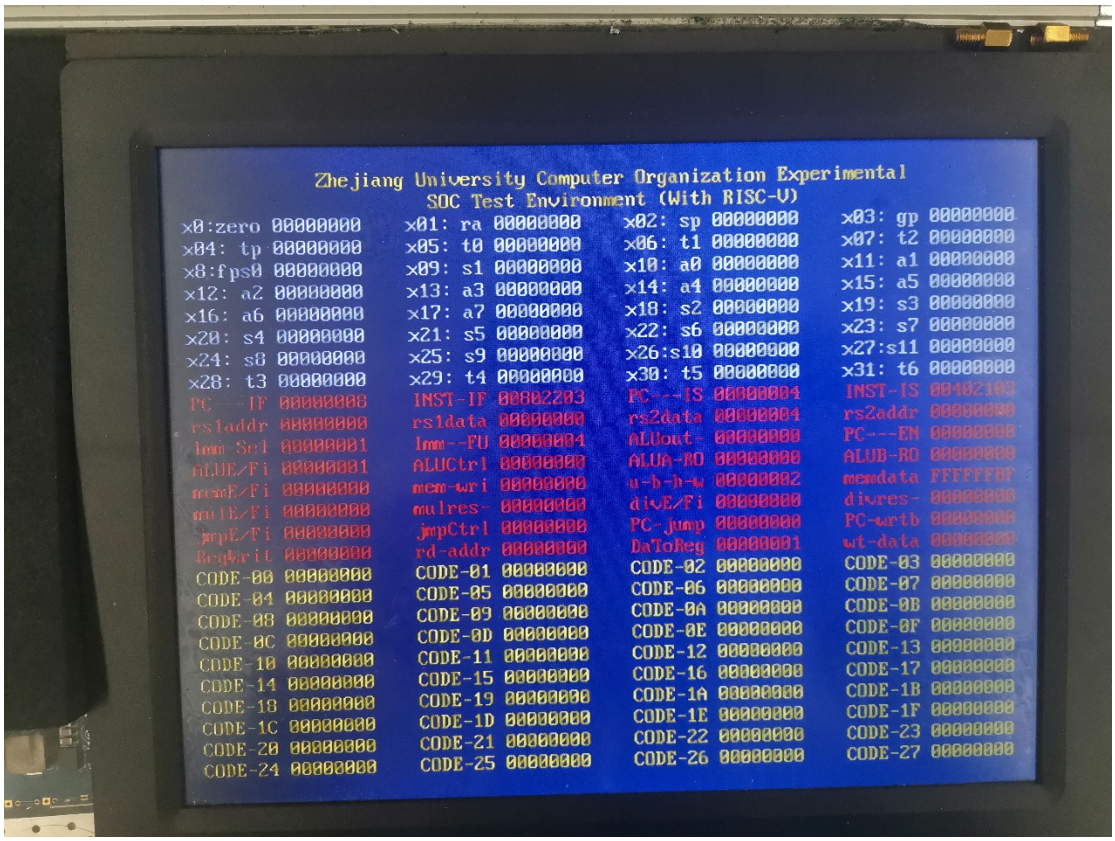
FU_mul：当 ID 阶段译码为乘法运算时，FU_mul_EN 被置为 1，之后使用六个周期进行相关操作，再进行下一条指令的执行。



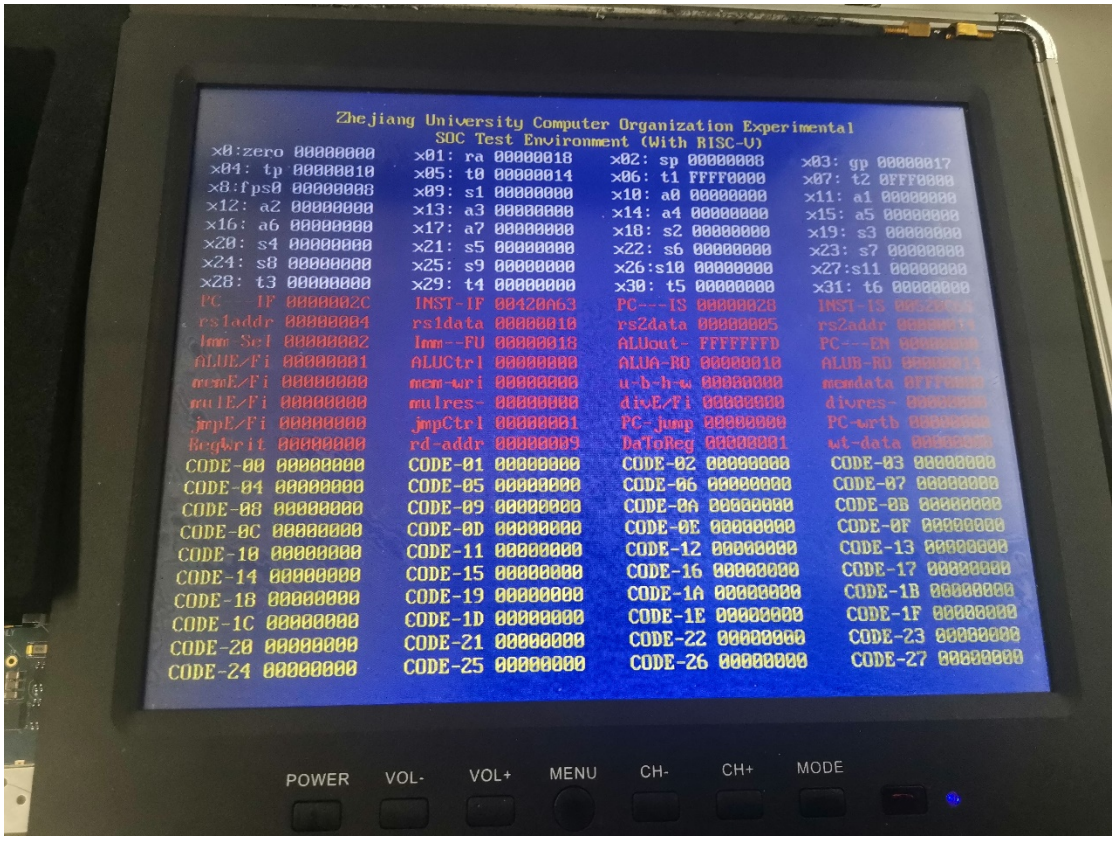
FU_div：当 ID 阶段译码为除法运算时，FU_div_EN 被置为 1，之后使用若干个周期进行相关操作，再进行下一条指令的执行。除法器运算的周期数会随输入变化。



实验箱结果：




Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000 x01: ra 00000000 x02: sp 00000008 x03: gp 00000000
x04: tp 00000000 x05: t0 00000000 x06: t1 00000000 x07: t2 00000000
x8:fps0 00000000 x09: s1 00000000 x10: a0 00000000 x11: a1 00000000
x12: a2 00000000 x13: a3 00000000 x14: a4 00000000 x15: a5 00000000
x16: a6 00000000 x17: a7 00000000 x18: s2 00000000 x19: s3 00000000
x20: s4 00000000 x21: s5 00000000 x22: s6 00000000 x23: s7 00000000
x24: s8 00000000 x25: s9 00000000 x26:s10 00000000 x27:s11 00000000
x28: t3 00000000 x29: t4 00000000 x30: t5 00000000 x31: t6 00000000
PC IF 00000010 INST-IF FFF08193 PC---IS 0000000C INST-IS 00410093
rs1addr 00000002 rs1data 00000008 rs2data 00000004 rs2addr 00000000
Imm Sel 00000000 Imm--FU 00000000 ALUout- 00000000 PC---EM 00000000
ALUE/Fi 00000000 ALUCtrl 00000001 ALUA-RO 00000008 ALUB-RO 00000000
memE/Fi 00000000 mem-wri 00000000 u-b-h-w 00000000 memdata 00000010
mulE/Fi 00000000 mulres- 00000000 divE/Fi 00000000 divres- 00000000
jmpE/Fi 00000000 jmpCtrl 00000000 PC-jump 00000008 PC-wrtb 00000000
RegWrit 00000000 rd-addr 00000004 DaToReg 00000002 wt-data 00000000
CODE-00 00000000 CODE-01 00000000 CODE-02 00000000 CODE-03 00000000
CODE-04 00000000 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000
CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000
CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000
CODE-10 00000000 CODE-11 00000000 CODE-12 00000000 CODE-13 00000000
CODE-14 00000000 CODE-15 00000000 CODE-16 00000000 CODE-17 00000000
CODE-18 00000000 CODE-19 00000000 CODE-1A 00000000 CODE-1B 00000000
CODE-1C 00000000 CODE-1D 00000000 CODE-1E 00000000 CODE-1F 00000000
CODE-20 00000000 CODE-21 00000000 CODE-22 00000000 CODE-23 00000000
CODE-24 00000000 CODE-25 00000000 CODE-26 00000000 CODE-27 00000000
POWER VOL- VOL+ MENU CH- CH+ MODE


Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000 x01: ra 00000018 x02: sp 00000008 x03: gp 00000017
x04: tp 00000010 x05: t0 00000014 x06: t1 FFFF0000 x07: t2 0FFF0000
x8:fps0 00000008 x09: s1 00000000 x10: a0 00000000 x11: a1 00000000
x12: a2 00000000 x13: a3 00000000 x14: a4 00000000 x15: a5 00000000
x16: a6 00000000 x17: a7 00000000 x18: s2 00000000 x19: s3 00000000
x20: s4 00000000 x21: s5 00000000 x22: s6 00000000 x23: s7 00000000
x24: s8 00000000 x25: s9 00000000 x26:s10 00000000 x27:s11 00000000
x28: t3 00000000 x29: t4 00000000 x30: t5 00000000 x31: t6 00000000
PC IF 0000002C INST-IF 00420A63 PC---IS 00000028 INST-IS 0052E733
rs1addr 00000004 rs1data 00000010 rs2data 00000005 rs2addr 00000013
Imm Sel 00000002 Imm--FU 00000018 ALUout- FFFFFFFD PC---EM 00000000
ALUE/Fi 00000001 ALUCtrl 00000000 ALUA-RO 00000010 ALUB-RO 00000013
memE/Fi 00000000 mem-wri 00000000 u-b-h-w 00000000 memdata 0FFF0000
mulE/Fi 00000000 mulres- 00000000 divE/Fi 00000000 divres- 00000000
jmpE/Fi 00000000 jmpCtrl 00000001 PC-jump 00000000 PC-wrtb 00000000
RegWrit 00000000 rd-addr 00000009 DaToReg 00000001 wt-data 00000000
CODE-00 00000000 CODE-01 00000000 CODE-02 00000000 CODE-03 00000000
CODE-04 00000000 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000
CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000
CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000
CODE-10 00000000 CODE-11 00000000 CODE-12 00000000 CODE-13 00000000
CODE-14 00000000 CODE-15 00000000 CODE-16 00000000 CODE-17 00000000
CODE-18 00000000 CODE-19 00000000 CODE-1A 00000000 CODE-1B 00000000
CODE-1C 00000000 CODE-1D 00000000 CODE-1E 00000000 CODE-1F 00000000
CODE-20 00000000 CODE-21 00000000 CODE-22 00000000 CODE-23 00000000
CODE-24 00000000 CODE-25 00000000 CODE-26 00000000 CODE-27 00000000
POWER VOL- VOL+ MENU CH- CH+ MODE


Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000 x01: ra 00000018 x02: sp 00000008 x03: gp 00000017
x04: tp 00000010 x05: t0 00000014 x06: t1 FFFF0000 x07: t2 0FFF0000
x8:fps0 00000008 x09: s1 FFFFFFFD x10: a0 00004000 x11: a1 00000048
x12: a2 FFFE0000 x13: a3 00000000 x14: a4 00000000 x15: a5 00000000
x16: a6 00000000 x17: a7 00000000 x18: s2 00000000 x19: s3 00000000
x20: s4 00000000 x21: s5 00000000 x22: s6 00000000 x23: s7 00000000
x24: s8 00000000 x25: s9 00000000 x26:s10 00000000 x27:s11 00000000
x28: t3 00000000 x29: t4 00000000 x30: t5 00000000 x31: t6 00000000
PC---IF 00000064 INST-IF 02268793 PC---IS 00000060
rs1addr 00000004 rs1data 00000018 rs2data 00000005
Imm-Sel 00000000 Imm--FU 00000000 ALUout- FFFE0000
ALUE/Fi 00000000 ALUCtrl 00000000 ALUA-RO 00000010
memE/Fi 00000000 mem-wri 00000000 u-b-h-w 00000000
mulE/Fi 00000000 mulres- 00000000 divE/Fi 00000000
jmpE/Fi 00000000 jmpCtrl 00000000 PC-jump 00000058
RegWrit 00000000 rd-addr 0000000D DaToReg 00000004
CODE-00 00000000 CODE-01 00000000 CODE-02 00000000 CODE-03 00000000
CODE-04 00000000 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000
CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000
CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000
CODE-10 00000000 CODE-11 00000000 CODE-12 00000000 CODE-13 00000000
CODE-14 00000000 CODE-15 00000000 CODE-16 00000000 CODE-17 00000000
CODE-18 00000000 CODE-19 00000000 CODE-1A 00000000 CODE-1B 00000000
CODE-1C 00000000 CODE-1D 00000000 CODE-1E 00000000 CODE-1F 00000000
CODE-20 00000000 CODE-21 00000000 CODE-22 00000000 CODE-23 00000000
CODE-24 00000000 CODE-25 00000000 CODE-26 00000000 CODE-27 00000000
POWER
VOL-
VOL+
MENU
CH-
CH+
MODE


Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000 x01: ra 00000018 x02: sp 00000008 x03: gp 00000017
x04: tp 00000010 x05: t0 00000014 x06: t1 FFFF0000 x07: t2 0FFF0000
x8:fps0 00000008 x09: s1 FFFFFFFD x10: a0 00004000 x11: a1 00000048
x12: a2 FFFE0000 x13: a3 01FFE000 x14: a4 00000000 x15: a5 00000000
x16: a6 00000000 x17: a7 00000000 x18: s2 00000000 x19: s3 00000000
x20: s4 00000000 x21: s5 00000000 x22: s6 00000000 x23: s7 00000000
x24: s8 00000000 x25: s9 00000000 x26:s10 00000000 x27:s11 00000000
x28: t3 00000000 x29: t4 00000000 x30: t5 00000000 x31: t6 00000000
PC---IF 00000068 INST-IF 00400813 PC---IS 00000064
rs1addr 0000000D rs1data 01FFE000 rs2data 00000002
Imm-Sel 00000000 Imm--FU 00000000 ALUout- FFFE0000
ALUE/Fi 00000000 ALUCtrl 00000000 ALUA-RO 01FFE000
memE/Fi 00000000 mem-wri 00000000 u-b-h-w 00000000
mulE/Fi 00000000 mulres- 00000140 divE/Fi 00000000
jmpE/Fi 00000000 jmpCtrl 00000000 PC-jump 00000058
RegWrit 00000000 rd-addr 0000000E DaToReg 00000003
CODE-00 00000000 CODE-01 00000000 CODE-02 00000000 CODE-03 00000000
CODE-04 00000000 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000
CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000
CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000
CODE-10 00000000 CODE-11 00000000 CODE-12 00000000 CODE-13 00000000
CODE-14 00000000 CODE-15 00000000 CODE-16 00000000 CODE-17 00000000
CODE-18 00000000 CODE-19 00000000 CODE-1A 00000000 CODE-1B 00000000
CODE-1C 00000000 CODE-1D 00000000 CODE-1E 00000000 CODE-1F 00000000
CODE-20 00000000 CODE-21 00000000 CODE-22 00000000 CODE-23 00000000
CODE-24 00000000 CODE-25 00000000 CODE-26 00000000 CODE-27 00000000

五、　讨论与心得

本次实验是在理解了多周期计算的流水线原理的基础上，设计多周期计算的流水线并对其进行验证。实验中已经提供了大部分的代码，只需要对 FU 相关的单元以及 RV32Core 进行填充，实验难度不高。实验主要是要求对于功能所需周期进行判断，然后使用 state 进行控制。在进行仿真验证时，需要比较小心注意每一个操作对应的波形，不然可能就会很难发现问题。RV32Core 的补充也一定要注意，不然可能会存在上板之后出现问题的可能。