# CODH - 流水线 CPU 设计　实验报告

院系:　　姓名:　　学号:

2023 年 7 月 13 日

## 1　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，特别是流水线中的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

## 2　实验环境

- macOS 13.0
- Rars1_5.jar (Riscv Assembler and Runtime Simulator)
- Vivado 2019.3
- Nexys 4 DDR 开发板

## 3　实验内容

### 3.1　流水线 CPU 数据通路的修改

- 添加级间寄存器存储不同阶段数据以及控制信号

### 3.2　流水线 CPU 相关及其处理

- 结构相关
- 数据相关
- 控制相关

之后将指令和数据存储器用 LabH3 中的文件初始化后，与串行调试单元模块（SDU_PL）整合，下载至 FPGA 中测试即可。

# 4 逻辑设计 / 核心代码

## 4.1 流水线 CPU 数据通路的修改

主要核心是四个级间寄存器的设计，它们是：

- IF/ID 寄存器
- ID/EX 寄存器
- EX/MEM 寄存器
- MEM/WB 寄存器

这些寄存器主要存储该阶段生成的需要传递到下一个阶段的数据，以及 PC/IR 值等。ID/EX 寄存器需要存储 EX 阶段、MEM 阶段、WB 阶段的控制信号，EX/MEM 寄存器需要存储 MEM 阶段、WB 阶段的控制信号，MEM/WB 寄存器需要存储 WB 阶段的控制信号。

具体设计可以参照 PPT 中的数据通路图，部分寄存器可能需要额外存储某些信息，这里不再赘述。

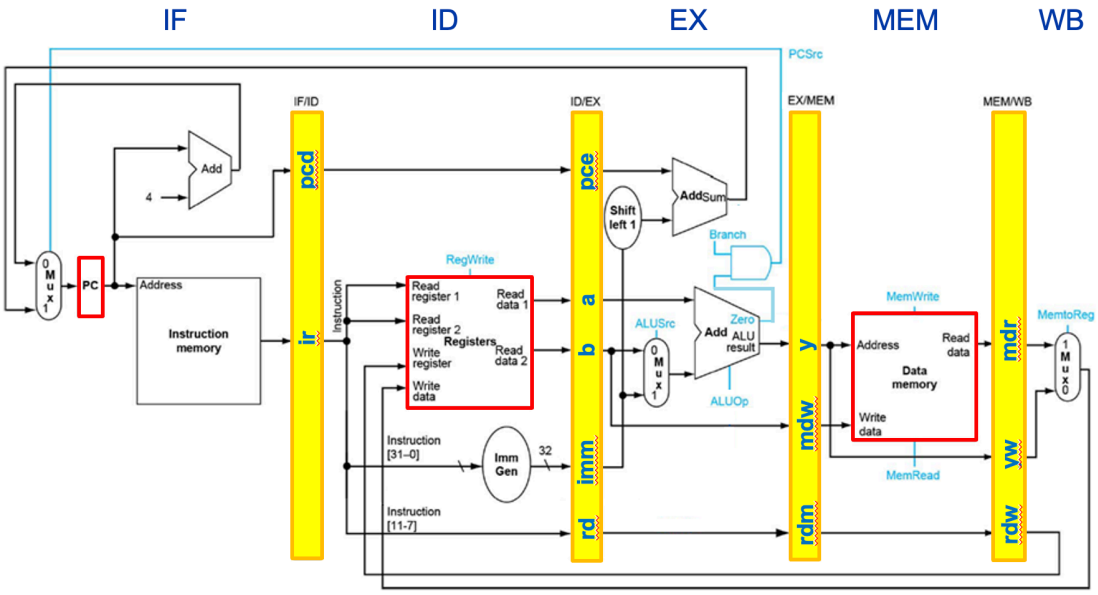


图 1: CPU-fig

下面列出这些寄存器的接口信息：

```
IF_ID_reg IF_ID_reg_inst(
    .clk(clk),
    .rstn(rstn),
    .stall(stall),
    .flush(flush),
    .pcd(IF_pc),
    .ir(IF_ir),
    ._pcd(ID_pc),
    ._ir(ID_ir),
    ._ra1(ID_rf_rs1),
    ._ra2(ID_rf_rs2),
    ._ire(ID_rf_rd)
);
```

```
14
15      ID_EX_reg ID_EX_reg_inst(
16          //input - ID
17          .clk(clk),
18          .rstn(rstn),
19          .stall(stall),
20          .flush(flush),
21          .alu_src(ID_alu_src),
22          .alu_op(ID_alu_op),
23          .pc_sel(ID_pc_sel),
24          .mem_write(ID_mem_write),
25          .reg_write(ID_reg_write),
26          .reg_sel(ID_reg_sel),
27          .pce(ID_pc),
28          .ir(ID_ir),
29          .a(ID_rf_rd1),
30          .b(ID_rf_rd2),
31          .rs1(ID_rf_rs1),
32          .rs2(ID_rf_rs2),
33          .imm(ID_imm),
34          .ire(ID_rf_rd),
35          //output - EX
36          ._EX(EX_EX),
37          .__mem_write(EX_mem_write),
38          ._WB(EX_WB),
39          .__pce(EX_pc),
40          .__ir(EX_ir),
41          .__a(EX_a),
42          .__b(EX_b),
43          .__imm(EX_imm),
44          .__ire(EX_rf_rd),
45          .__rs1(EX_rf_rs1),
46          .__rs2(EX_rf_rs2)
47      );
48
49      EX_MEM_reg EX_MEM_reg_inst (
50          //-----INPUT-----
51          .clk(clk),
52          .rstn(rstn),
53          .mem_write(EX_mem_write),
54          .WB(EX_WB),
55          .imm(EX_imm),
56          .pcm(EX_pc),
57          .y(EX_alu_out),
58          .mdw(EX_b),
59          .irm(EX_rf_rd),
60          .ir(EX_ir),
61          .opcode(EX_opcode),
62          //-----OUTPUT-----
63          //control
64          .__mem_write(MEM_mem_write),
65          ._WB(MEM_WB),
66          //data
67          .__y(MEM_addr),
68          .__pcm(MEM_pc),
69          .__imm(MEM_imm),
70          .__mdw(MEM_d),
71          .__irm(MEM_rf_rd),
72          .__ir(MEM_ir),
```

```
73          .__opcode(MEM__opcode)
74      );
75
76      MEM_WB_reg MEM__WB_reg_inst(
77          //------INPUT------
78          .clk(clk),
79          .rstn(rstn),
80          .WB(MEM_WB),
81          .__imm(MEM_imm),
82          .pcw(MEM_pc),
83          .mdr(MEM_dm_rd),
84          .vw(MEM_addr),
85          .irw(MEM_rf_rd),
86          .ir(MEM_ir),
87          .opcode(MEM_opcode),
88          //------OUTPUT------
89          //control
90          .__reg__write(WB__reg__write),
91          .__reg__sel(WB__reg__sel),
92          //data
93          .__mdr(WB_data),
94          .__vw(WB_vw),
95          .__irw(WB__rf_rd),
96          .__pcw(WB_pc),
97          .___imm(WB_imm),
98          .__ir(WB__ir),
99          .__opcode(WB__opcode)
100     );
```

cpu.v

在 ID 阶段产生控制信号后，通过寄存器传递至后面的阶段即可利用，至此便得到了在非特殊情况下可正常运作的流水线 CPU

## 4.2 流水线 CPU 相关及其处理

具体有三种相关需要特殊考虑，否则由于流水线 CPU 的特性不能正常运行。

### 4.2.1 结构相关

多条指令可能会同时访问一个存储。而一个存储器同一时钟周期只能写一次，考虑到这一点，我们采用哈佛结构，将指令存储器和数据存储器区分开来，这样便解决了端口冲突的问题。

另外有可能在 WB 阶段末尾需要写回的寄存器在下一个时钟周期的 ID 阶段就要读取。由于是时钟上升沿才进行写回，所以需要将寄存器改为写优先的读方式来解决这一问题，如下：

```
1   assign rd1 = (ra1 == wa && we && wa != 0) ? wd : rf[ra1];
```

cpu.v

### 4.2.2 数据相关

这一般出现在次条指令需要用到前面指令的运算结果时，分为两种情况：

1. 前一条指令是 LOAD 指令之外的指令。次条指令段最晚在 EX 阶段需要将 alu_in 的值替换为前一条指令的运算结果，此时上一条指令（或者更早）恰好执行到了 MEM/WB 阶段，可以将此阶段寄存器读出的运算结果 forward 至下一条指令的 EX 阶段输入即可。

2. 前一条指令是 LOAD 指令，则 LOAD 的结果至少在 MEM 阶段结束（WB 阶段开始时）时才可用 forward 到上一条指令的 EX 阶段。因此需要阻止紧随 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）。

```
    forward forward_inst(
        .EX_rs1(EX_rf_rs1),
        .EX_rs2(EX_rf_rs2),
        .MEM_rd(MEM_rf_rd),
        .WB_rd(WB_rf_rd),
        .MEM_reg_write(MEM_WB[3]),
        .WB_reg_write(WB_reg_write),
        .MEM_reg_sel(MEM_WB[2:0]),
        .WB_reg_sel(WB_reg_sel),
        .EX_a(EX_a),
        .EX_b(EX_b),
        .MEM_alu_out(MEM_addr),
        .WB_rf_wd(WB_rf_wd),
        .alu_in1(a_forward),
        .alu_in2(b_forward),
        .MEM_pc(MEM_pc),
        .MEM_dm_rd(MEM_dm_rd),
        .MEM_imm(MEM_imm)
    );
```

cpu.v

具体的实现方式是编写 forward 模块，如果识别到 EX 阶段所读的寄存器地址恰好是 MEM 阶段/WB 阶段的目标寄存器地址，则用 MEM 阶段/WB 阶段的运算结果提前替换 EX 阶段的 alu_in 值，否则不做任何操作。

而对于紧随 LOAD 指令的指令，需要让其暂停一个时钟周期，这里采用编写 stall 模块（实际把 stall 和后文的 flush 模块合并在一起了），如果识别到 EX 阶段所读的寄存器地址恰好是上一条指令 MEM 阶段的目标寄存器地址，且上一条指令恰好为 LOAD，则将 stall 信号置为 1，否则为 0。级间寄存器若识别到 stall 信号为 1，则不在时钟上升沿写入数据，而是将数据保持不变，这样便实现了暂停一个时钟周期的功能。

#### 4.2.3 控制相关

这方面的问题主要是由于控制相关指令决定跳转与否至少需要到 EX 阶段才能确定，而此时 ID 阶段已经读取了下一条指令。因此需要根据 branch 信号以及 EX 阶段的 opcode 来判断当前指令是否会产生跳转，若是，则当前 IF/ID 阶段按顺序读取的两条指令都是错误的（应该改为正确的跳转后的指令），因此置 flush 信号为 1。级间寄存器若识别到 flush 信号为 1，则不在时钟上升沿写入数据，而是将数据清零，这样便实现了清空 IF/ID 寄存器，ID/EX 寄存器的功能，使得错误的指令不会进入流水线。

```
    stall_flush stall_flush_inst(
        .branch(branch),
        .opcode(EX_opcode),
        .ire(EX_rf_rd),
```

```
        .rf_rs1(ID_rf_rs1),
        .rf_rs2(ID_rf_rs2),
        .stall(stall),
        .flush(flush)
    );
```

cpu.v

```
    always@(*) begin
        if ((opcode == I_LOAD) && ((ire == rf_rs1) || (ire == rf_rs2)))
            stall = 1;
        else
            stall = 0;

        if (branch || (opcode == J_JAL) || (opcode == I_JALR)) begin
            flush = 1;
        end
        else begin
            flush = 0;
        end
    end
```

stall_flush.v

烧录后测试程序以及排序程序的正确性已经由助教检查完成，流水线 CPU 设计任务至此完成。

### 4.2.4 结果分析

RTL 电路：

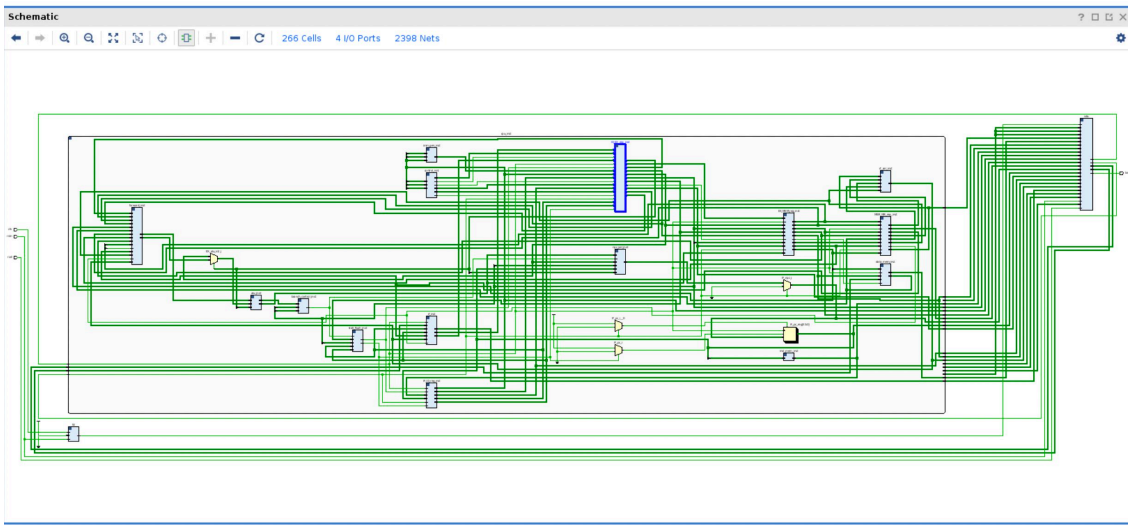


图 2: RTL

电路资源使用情况：

Hierarchy

| Name | Slice LUTs (63400) | Slice Registers (126800) | F7 Muxes (31700) | F8 Muxes (15850) | Bonded IOB (210) | BUFGCTRL (32) |
|---|---|---|---|---|---|---|
| MAIN | 3836 | 2087 | 1002 | 204 | 4 | 7 |
| cpu_inst (cpu) | 3240 | 1822 | 960 | 192 | 0 | 0 |
| control_isnt ( | 7 | 7 | 0 | 0 | 0 | 0 |
| data_mem_ins | 1160 | 64 | 576 | 0 | 0 | 0 |
| EX_MEM_reg_i | 151 | 170 | 0 | 0 | 0 | 0 |
| forward_inst ( | 261 | 64 | 0 | 0 | 0 | 0 |
| ID_EX_reg_ins | 410 | 189 | 0 | 0 | 0 | 0 |
| IF_ID_reg_inst | 130 | 71 | 0 | 0 | 0 | 0 |
| inst_mem_inst | 85 | 0 | 0 | 0 | 0 | 0 |
| MEM_WB_reg_ | 153 | 169 | 0 | 0 | 0 | 0 |
| npc_sel_inst ( | 1 | 32 | 0 | 0 | 0 | 0 |
| rf_inst (reg_fil | 864 | 992 | 384 | 192 | 0 | 0 |
| rf_sel_inst (re | 0 | 32 | 0 | 0 | 0 | 0 |
| fd (udf_FD) | 24 | 33 | 0 | 0 | 0 | 0 |
| sdu (SDU) | 572 | 232 | 42 | 12 | 0 | 0 |

图 3: UTL1

电路性能:

| Name | Slack | Levels | Routes | High Fanout | From | To | Total Delay | Logic Delay | Net Delay | Requirement | Source Clock | Destination C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Path 1 | 9.613 | 10 | 11 | 5 | fd/counter_reg[2]/C | fd/counter_reg[0]/D | 5.251 | 2.606 | 2.645 | 15.0 | sys_clk_pin | sys_clk_pin |
| Path 2 | 9.613 | 10 | 11 | 5 | fd/counter_reg[2]/C | fd/counter_reg[1]/D | 5.251 | 2.606 | 2.645 | 15.0 | sys_clk_pin | sys_clk_pin |
| Path 3 | 9.613 | 10 | 11 | 5 | fd/counter_reg[2]/C | fd/counter_reg[3]/D | 5.251 | 2.606 | 2.645 | 15.0 | sys_clk_pin | sys_clk_pin |
| Path 4 | 9.613 | 10 | 11 | 5 | fd/counter_reg[2]/C | fd/counter_reg[7]/D | 5.251 | 2.606 | 2.645 | 15.0 | sys_clk_pin | sys_clk_pin |
| Path 5 | 9.613 | 10 | 11 | 5 | fd/counter_reg[2]/C | fd/counter_reg[9]/D | 5.251 | 2.606 | 2.645 | 15.0 | sys_clk_pin | sys_clk_pin |
| Path 6 | 10.807 | 4 | 5 | 33 | fd/counter_reg[23]/C | fd/counter_reg[0]/CE | 3.811 | 1.145 | 2.666 | 15.0 | sys_clk_pin | sys_clk_pin |
| Path 7 | 10.807 | 4 | 5 | 33 | fd/counter_reg[23]/C | fd/counter_reg[10]/CE | 3.811 | 1.145 | 2.666 | 15.0 | sys_clk_pin | sys_clk_pin |
| Path 8 | 10.807 | 4 | 5 | 33 | fd/counter_reg[23]/C | fd/counter_reg[11]/CE | 3.811 | 1.145 | 2.666 | 15.0 | sys_clk_pin | sys_clk_pin |
| Path 9 | 10.807 | 4 | 5 | 33 | fd/counter_reg[23]/C | fd/counter_reg[12]/CE | 3.811 | 1.145 | 2.666 | 15.0 | sys_clk_pin | sys_clk_pin |
| Path 10 | 10.807 | 4 | 5 | 33 | fd/counter_reg[23]/C | fd/counter_reg[13]/CE | 3.811 | 1.145 | 2.666 | 15.0 | sys_clk_pin | sys_clk_pin |

图 4: TIMING

# 5 实验总结

1. 通过本次实验，我学习到了如何依据需求设计流水线 CPU 的数据通路，也就是如何修改单周期 CPU 的数据通路
2. 通过本次实验，我理解了 CPU 各阶段之间的数据依赖关系以及控制信号关系
3. 通过本次实验，我了解了流水线 CPU 相较于但周期 CPU 可能出现的不稳定因素，例如数据相关、控制相关等，以及如何通过设置相关模块机制（forward、flush、stall）解决这些问题
4. 通过本次实验，我充分体会到了命名清晰的重要性，花费一定时间修改命名也可以让后续开发更加顺利
5. 最重要的是，在本次试验中，所有编写调试编译生成操作全是在 M1 芯片平台上进行的。经过漫长的两个月后，终于可以在 M1 芯片上编译运行代码调试开发板了，这是一个值得庆祝的事情。

# 6 意见/建议

无，这个实验设计很完美。希望实验文档再详细一些就好了