中国科学技术大学
University of Science and Technology of China

计算机组成原理

Lab5 流水线CPU设计

计算机实验教学中心

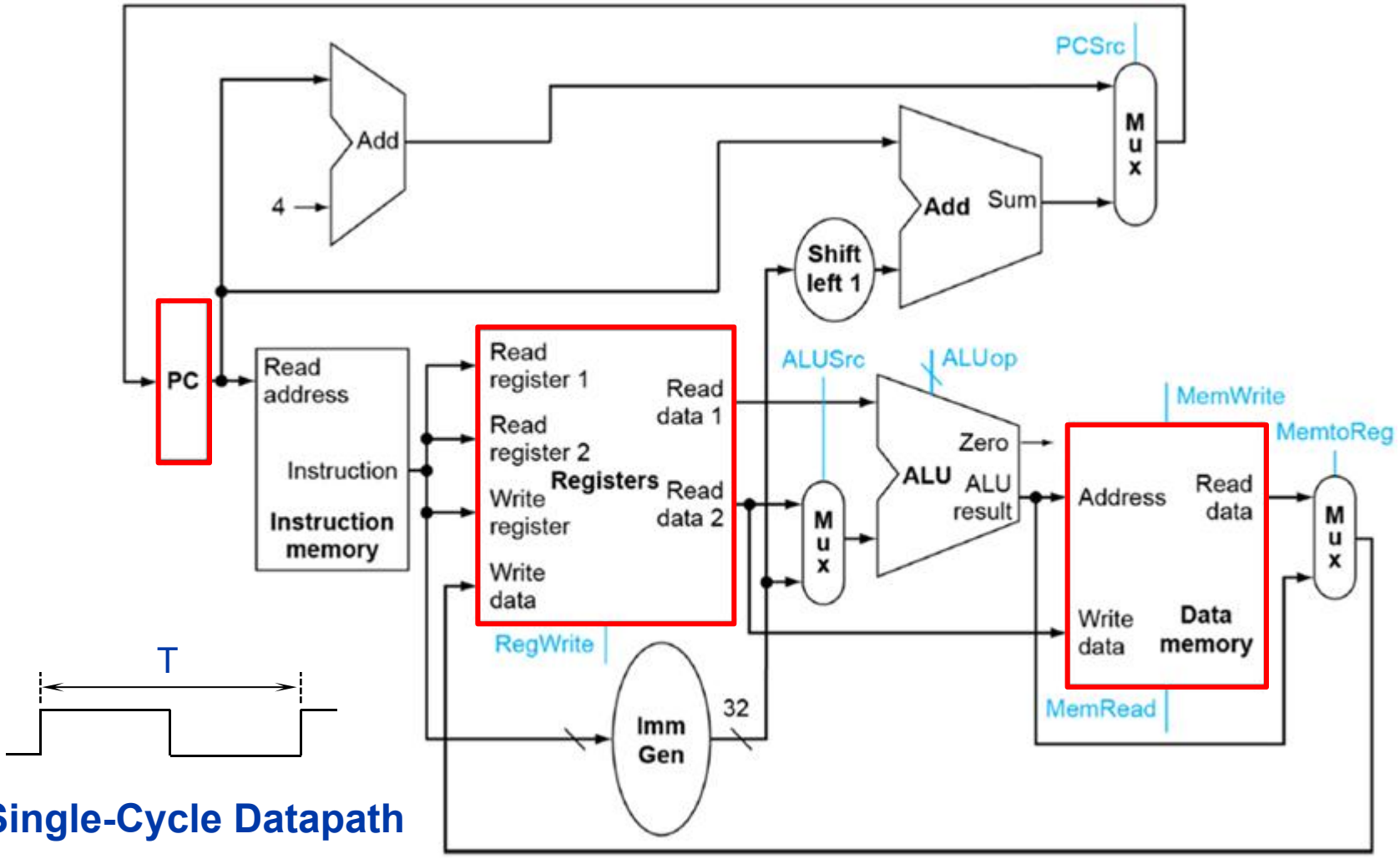
2023-5-8

实验目标

- 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

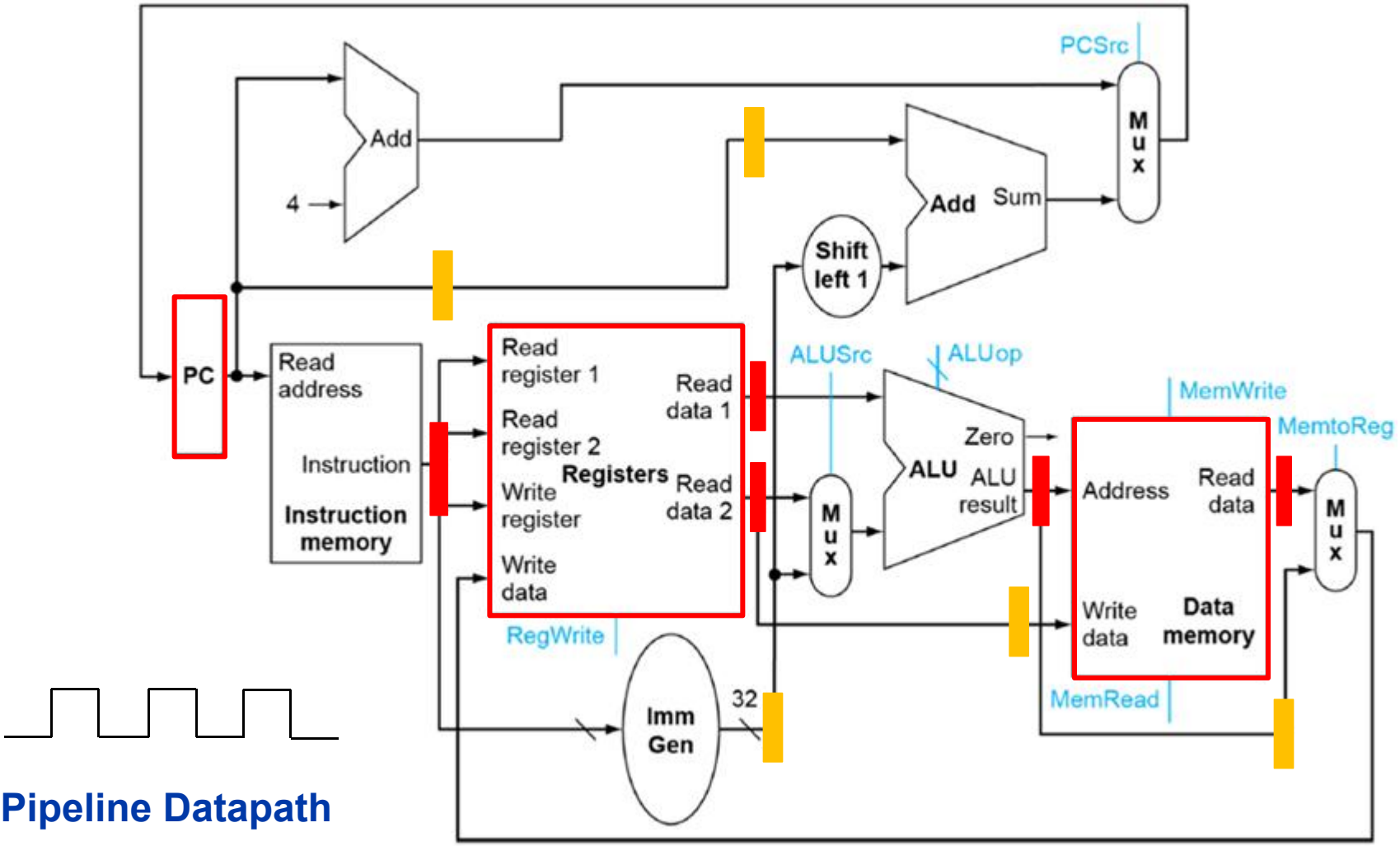
1. 数据通路_单周期CPU



Single-Cycle Datapath

实验原理

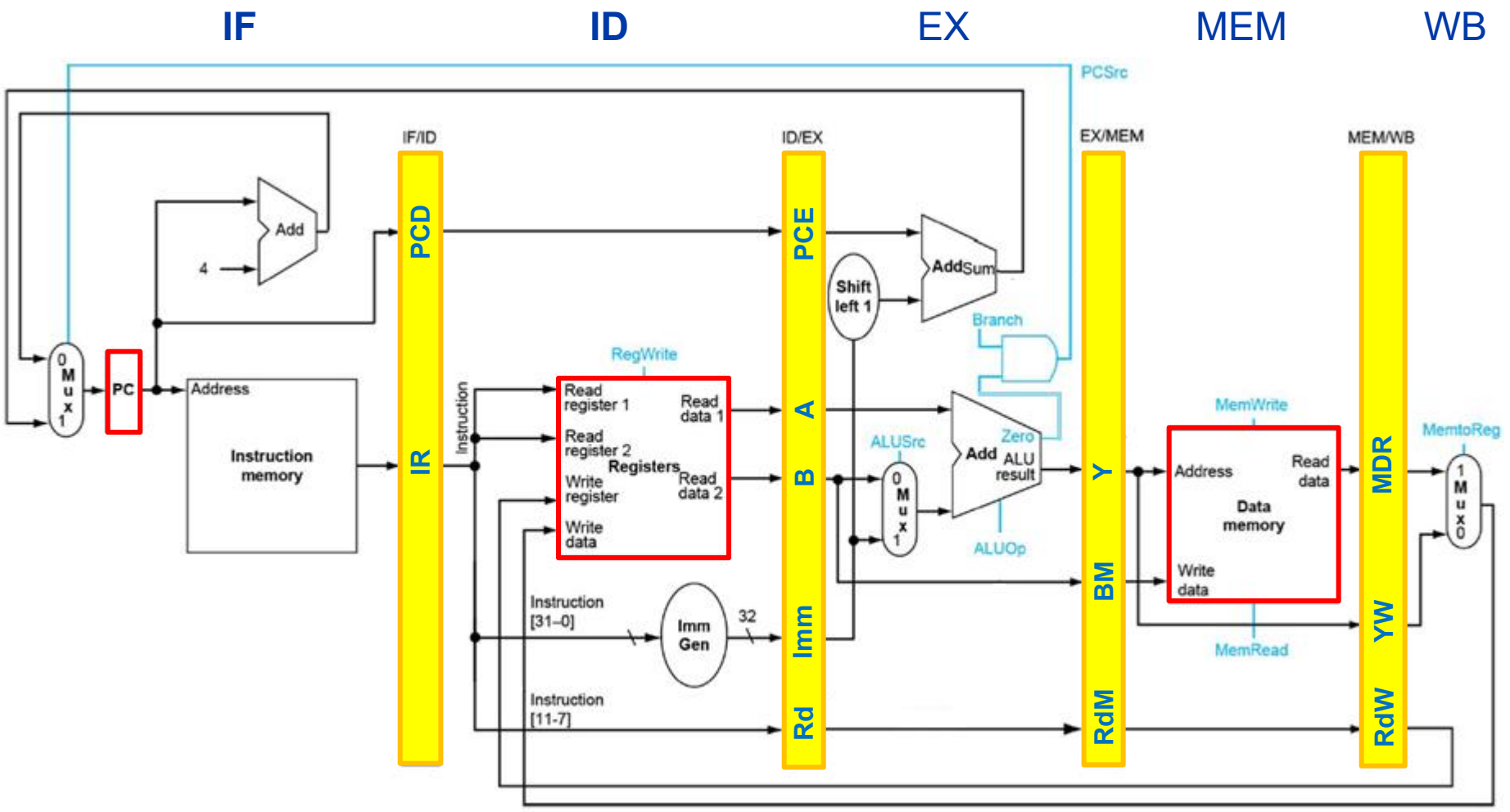
1. 数据通路_流水线CPU



Pipeline Datapath

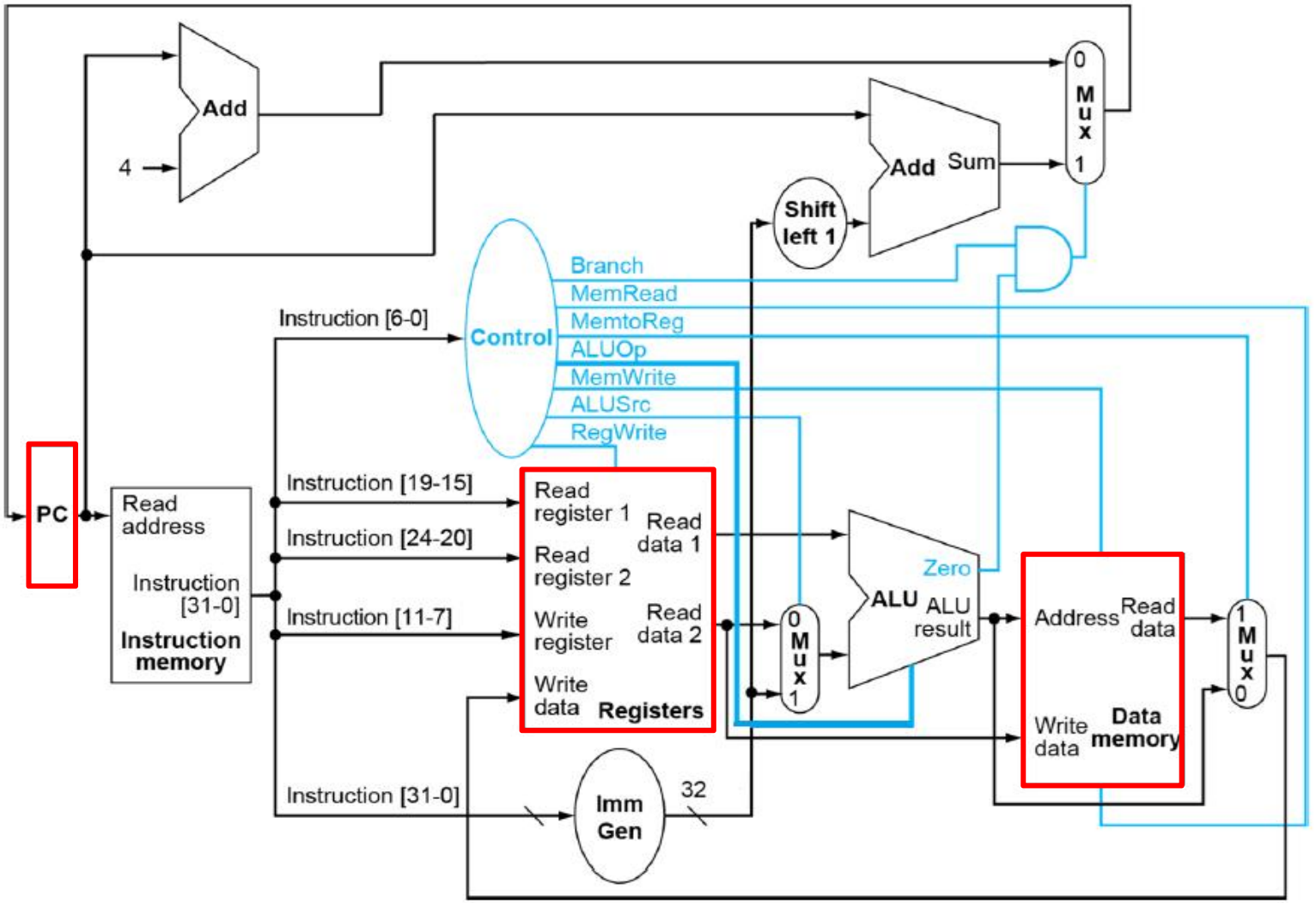
实验原理

1. 数据通路_流水线CPU



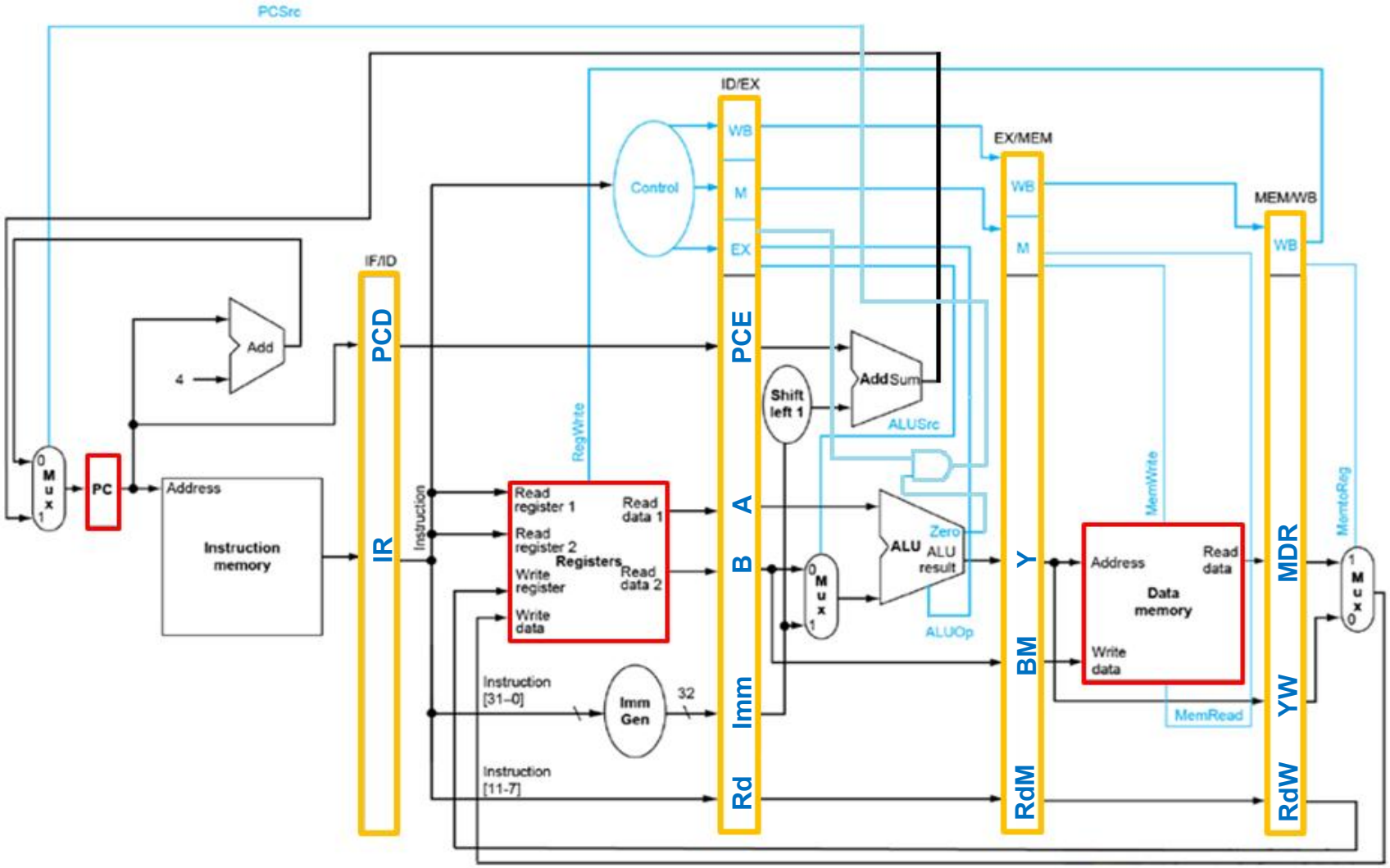
实验原理

2. 数据通路+控制器_单周期CPU



实验原理

2. 数据通路+控制器_流水线CPU



实验原理

3. 流水线相关及其处理

□ 结构相关：当多条指令执行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

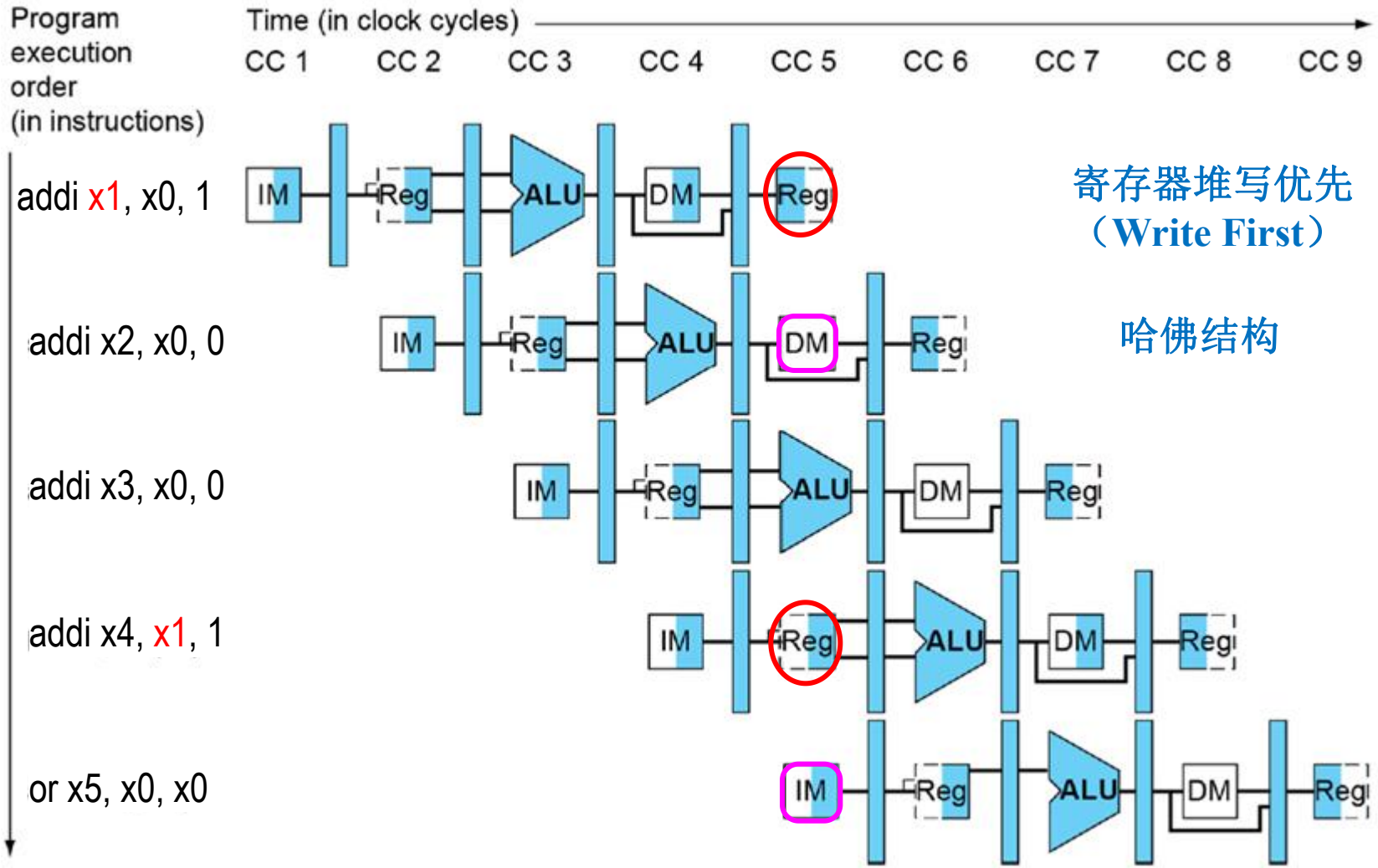
- ✓ 数据定向（Forwarding）：将执行结果提前传递至之前流水段
- ✓ 加载-使用相关（Load-use hazard）：阻止紧随Load已进入流水线的指令流动（Stall），向后续流水段插入空操作（Bubble）

□ 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

实验原理

3. 流水线相关及其处理_结构相关

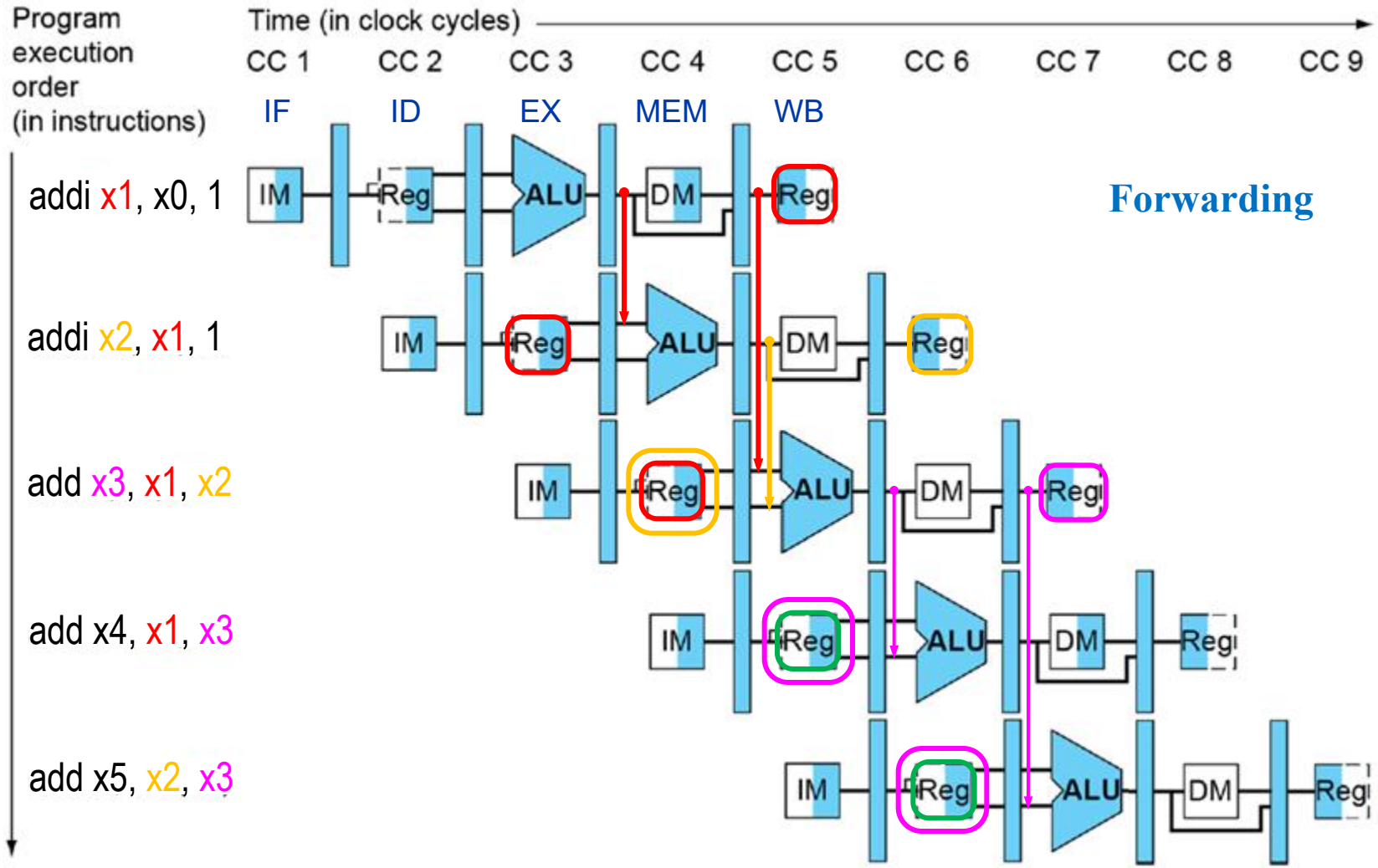


寄存器堆写优先（Write First）

哈佛结构

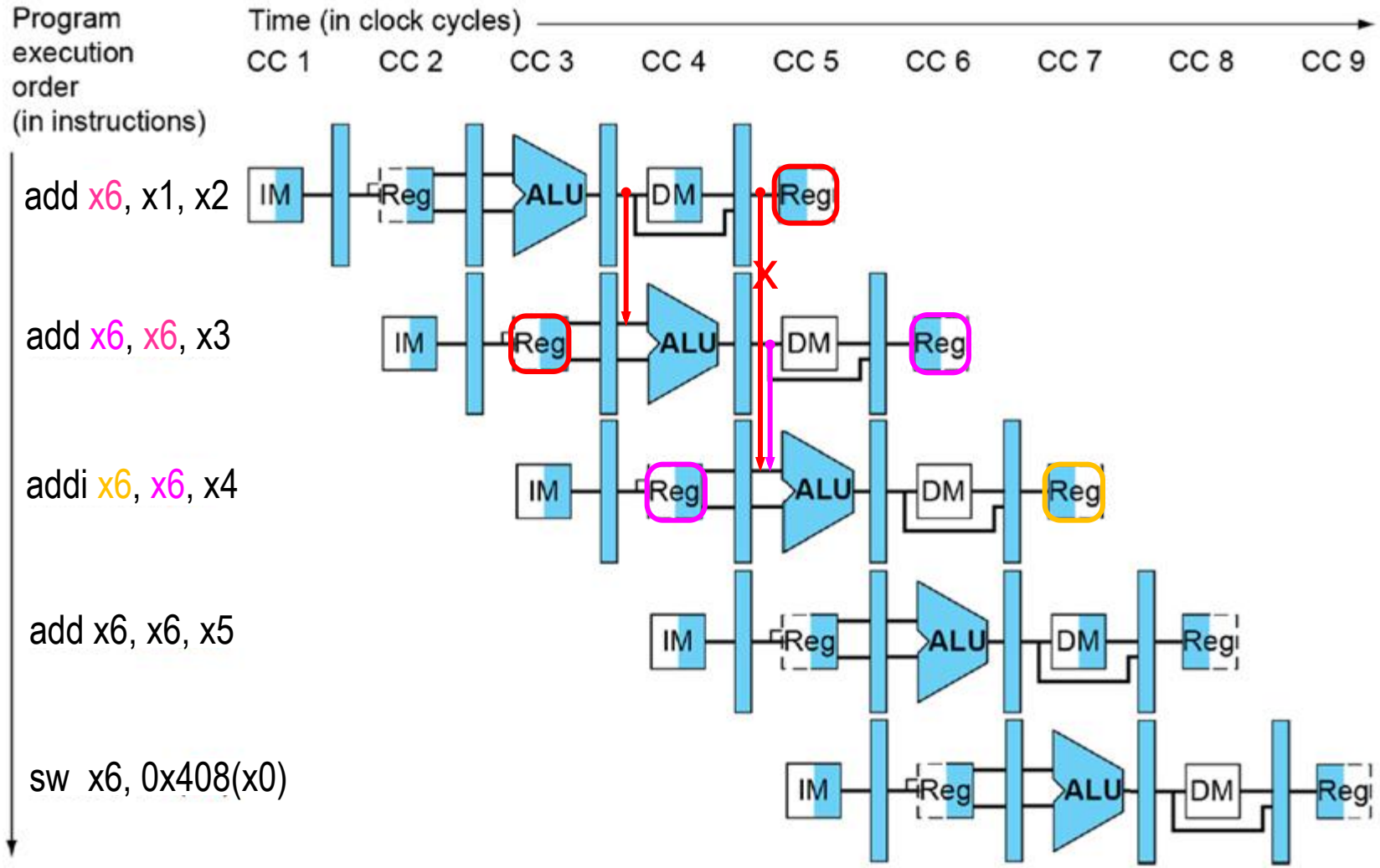
实验原理

3. 流水线相关及其处理_数据相关



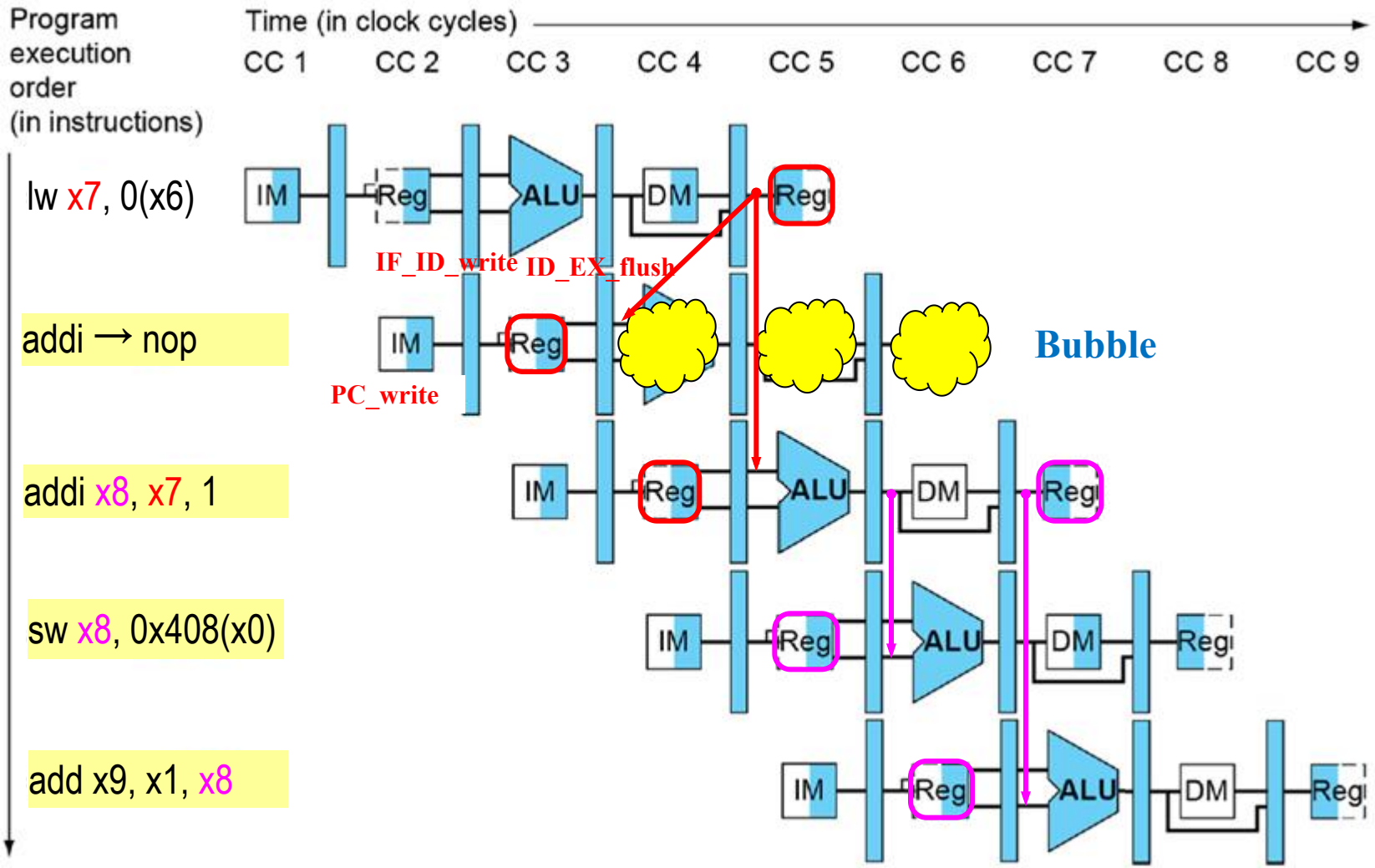
实验原理

3. 流水线相关及其处理_数据相关



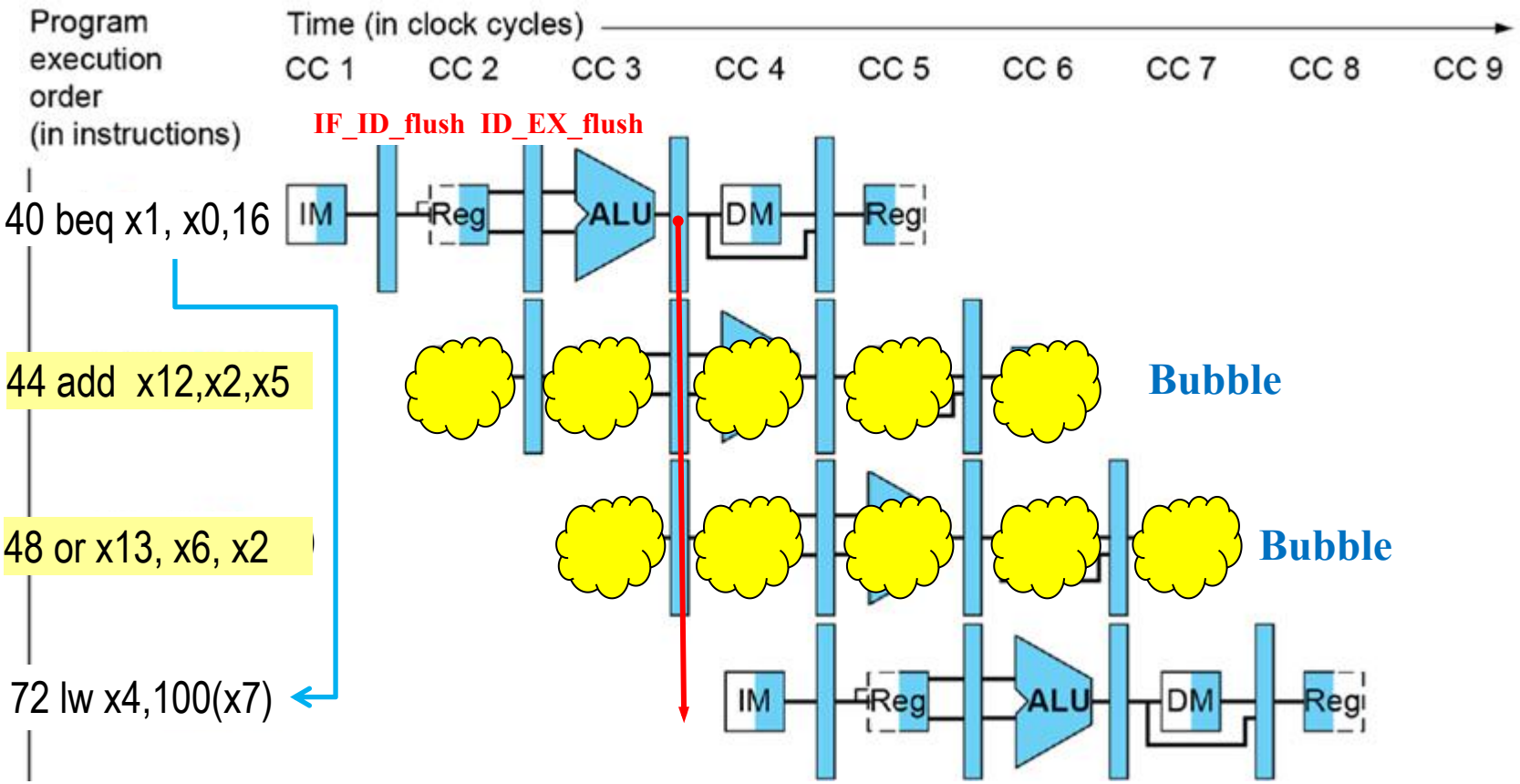
实验原理

3. 流水线相关及其处理_数据相关



实验原理

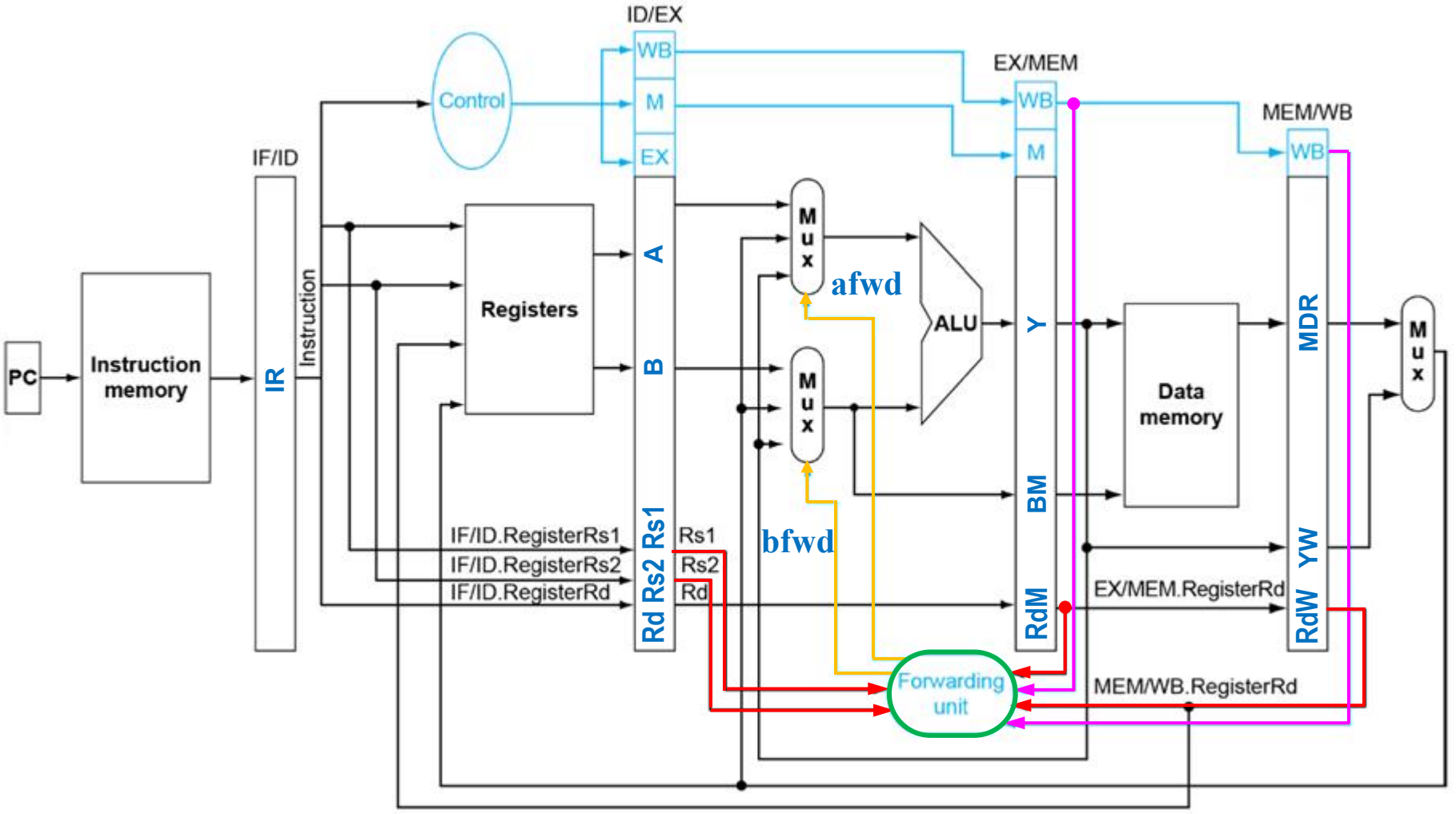
3. 流水线相关及其处理_控制相关



<注意段间寄存器设计时采用同步清空:IF_ID_flush,ID_EX_flush>

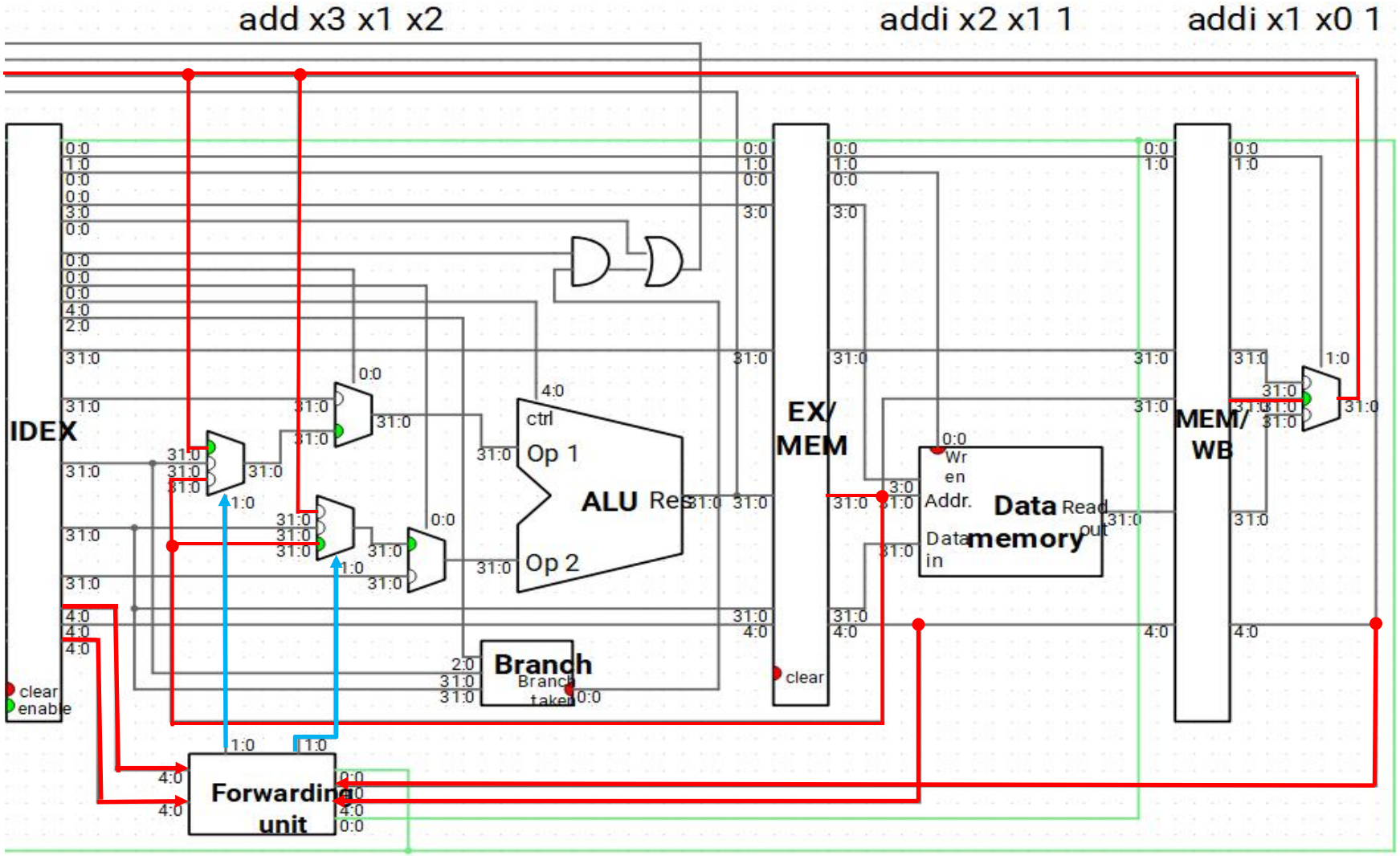
实验原理

3. 流水线相关及其处理_Forwarding



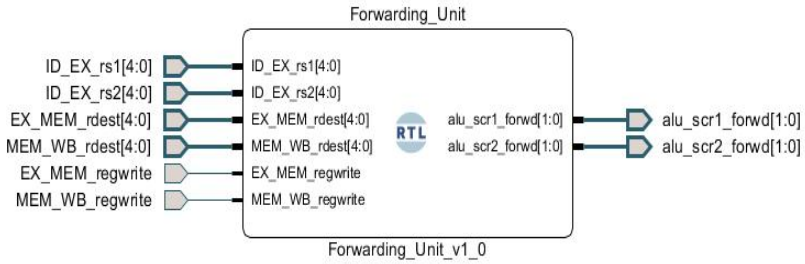
实验原理

3. 流水线相关及其处理_Forwarding(Ripes)



实验原理

3. 流水线相关及其处理_Forwarding(参考设计)



第一种数据相关：寄存器堆写回指令+其他指令

1）EXE段数据冲突检测条件：

If (EX/MEM.RegWrite and (EX/MEM.RegisterRd/=0) and (EX/MEM.RegisterRd==ID/EX.RegisterRs1))

ForwardA=01

If (EX/MEM.RegWrite and (EX/MEM.RegisterRd/=0) and (EX/MEM.RegisterRd==ID/EX.RegisterRs2))

ForwardB=01

MEM段数据冲突检测条件：

If (MEM/WB.RegWrite and (MEM/WB.RegisterRd/=0) and (MEM/WB.RegisterRd==ID/EX.RegisterRs1))

ForwardA=10

If (MEM/WB.RegWrite and (MEM/WB.RegisterRd/=0) and (MEM/WB.RegisterRd==ID/EX.RegisterRs2))

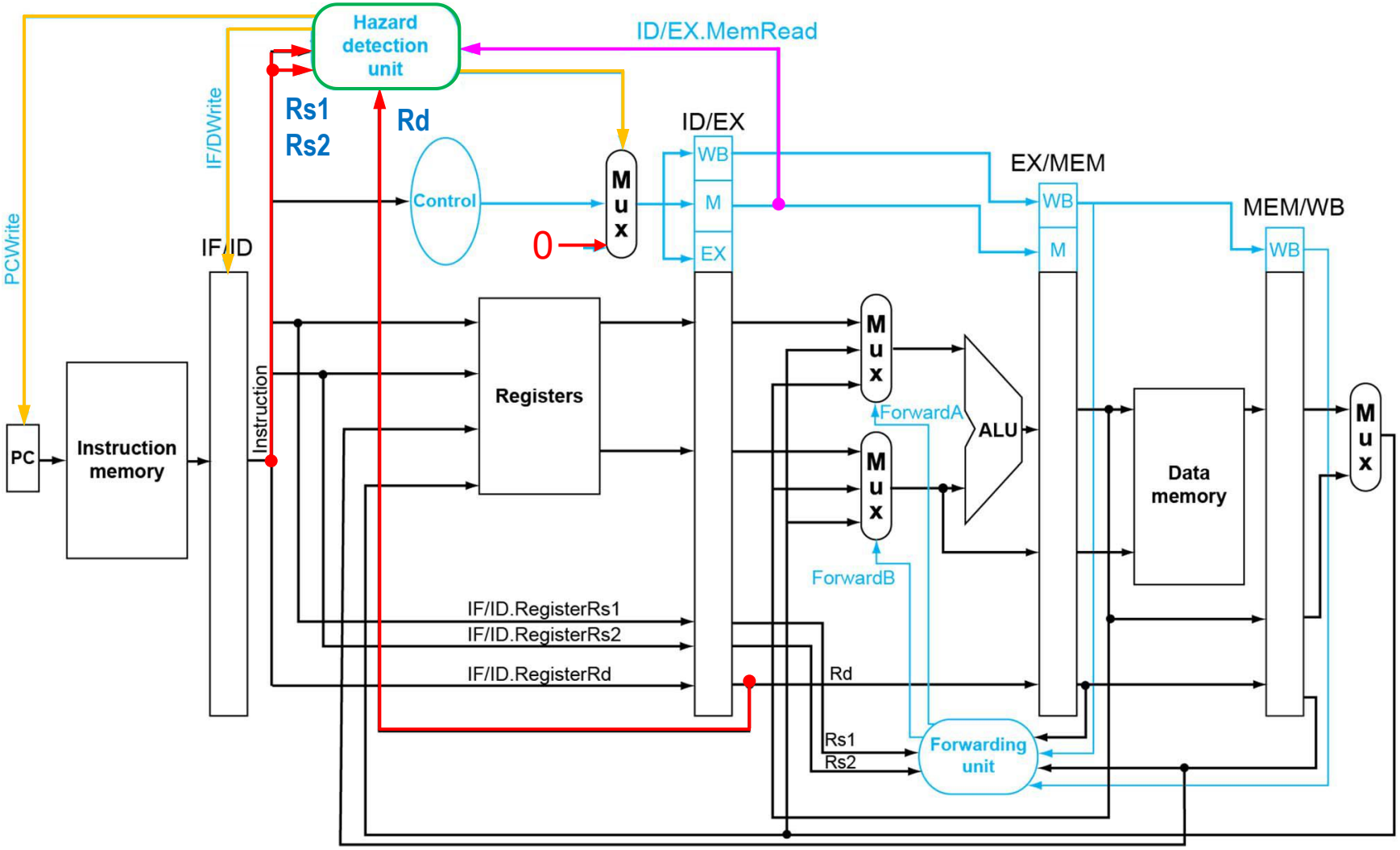
ForwardB=10

2）解决方案：增加旁路（ forwarding unit）

```
module Forwarding_Unit(
    input [4:0] ID_EX_rs1,
    input [4:0] ID_EX_rs2,
    input [4:0] EX_MEM_rdest,
    input [4:0] MEM_WB_rdest,
    input EX_MEM_regwrite,
    input MEM_WB_regwrite,
    output reg [1:0] alu_scr1_forwd,
    output reg [1:0] alu_scr2_forwd
    );

    always @(*)
    begin
    if( EX_MEM_regwrite && (EX_MEM_rdest != 5'd0) && (EX_MEM_rdest == ID_EX_rs1))
        alu_scr1_forwd = 2'b01;
    else if( MEM_WB_regwrite && (MEM_WB_rdest != 5'd0) && (MEM_WB_rdest == ID_EX_rs1))
        alu_scr1_forwd = 2'b10;
    else alu_scr1_forwd = 2'b00;
    end

    always @(*)
    begin
    if( EX_MEM_regwrite && (EX_MEM_rdest != 5'd0) && (EX_MEM_rdest == ID_EX_rs2))
        alu_scr2_forwd = 2'b01;
    else if( MEM_WB_regwrite && (MEM_WB_rdest != 5'd0) && (MEM_WB_rdest == ID_EX_rs2))
        alu_scr2_forwd = 2'b10;
    else alu_scr2_forwd = 2'b00;
    end


endmodule
```

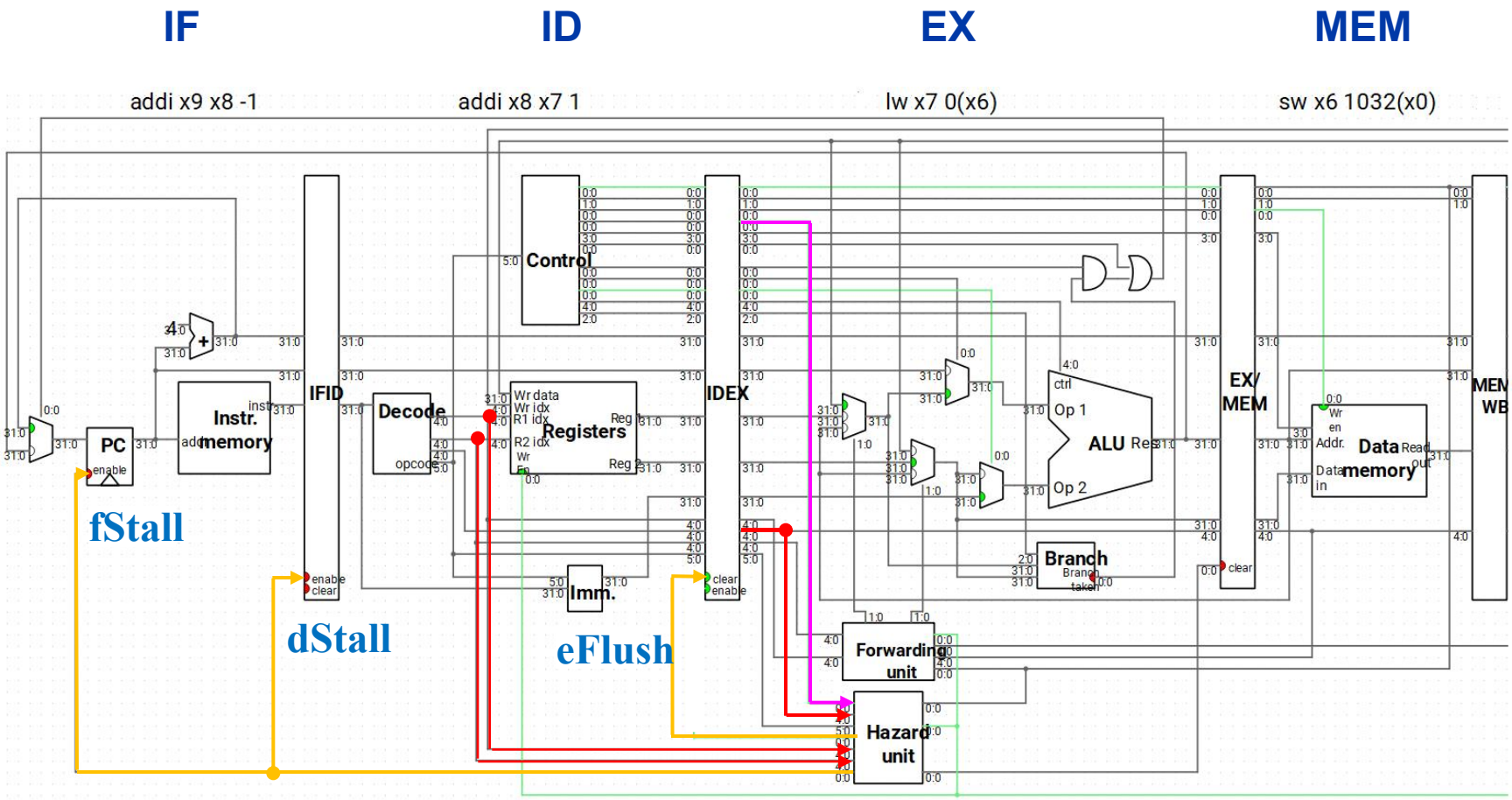
实验原理

3. 流水线相关及其处理_Load-Use Hazard



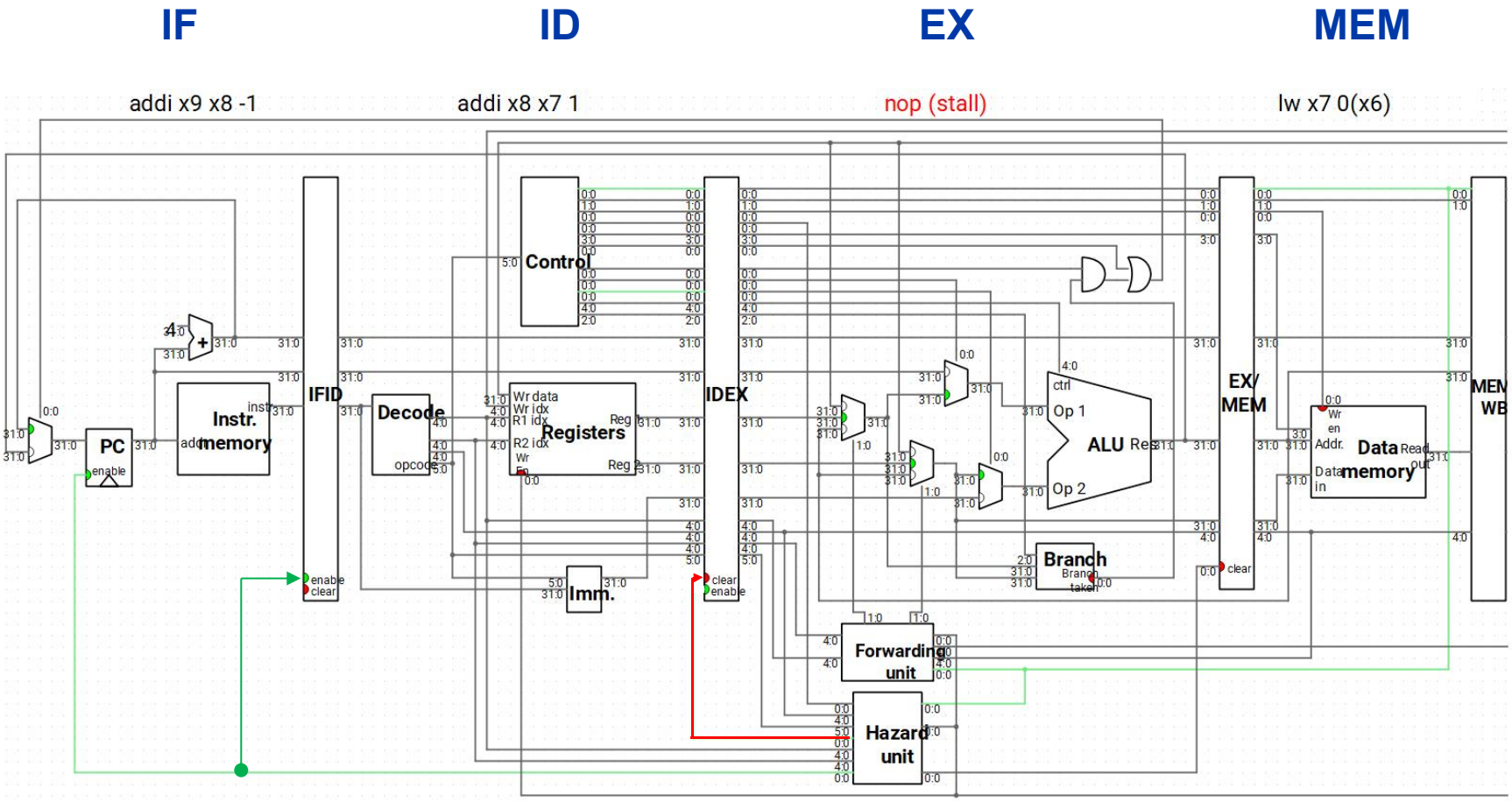
实验原理

3. 流水线相关及其处理_Load-Use Hazard (Ripes)



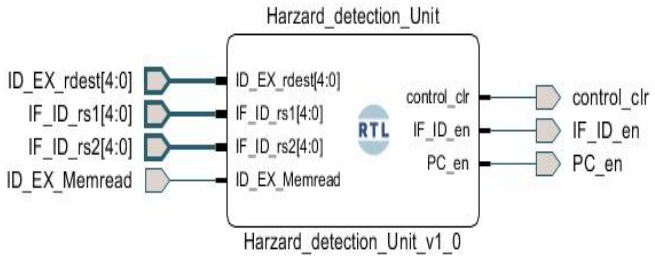
实验原理

3. 流水线相关及其处理_Load-Use Hazard (Ripes)



实验原理

3. 流水线相关及其处理_Load-Use Hazard (参考设计)



第二种数据相关：LW指令+其他指令

1）冲突检测条件：

a)上一条指令是Load指令（特征：ID/EXE.Memread）

b)上一条Load指令的写入目的寄存器和当前指令的某一源寄存器相同

ID/EX.Memread and

(ID/EX.registerRd/=0)and(ID/EX.registerRd==IF/ID.registerRs1 or ID/EX.registerRd==IF/ID.registerRs2)

2）解决方案： Harzard_detection_Unit

a)让当前指令的控制信号全部为0（bubble down,插入气泡）

b)让PC值保持不变(阻止更新PC,Freeze up，使之重复)

c)让IF/ID段寄存器保持不变（阻止更新IF/ID，Freeze up，使之重复）

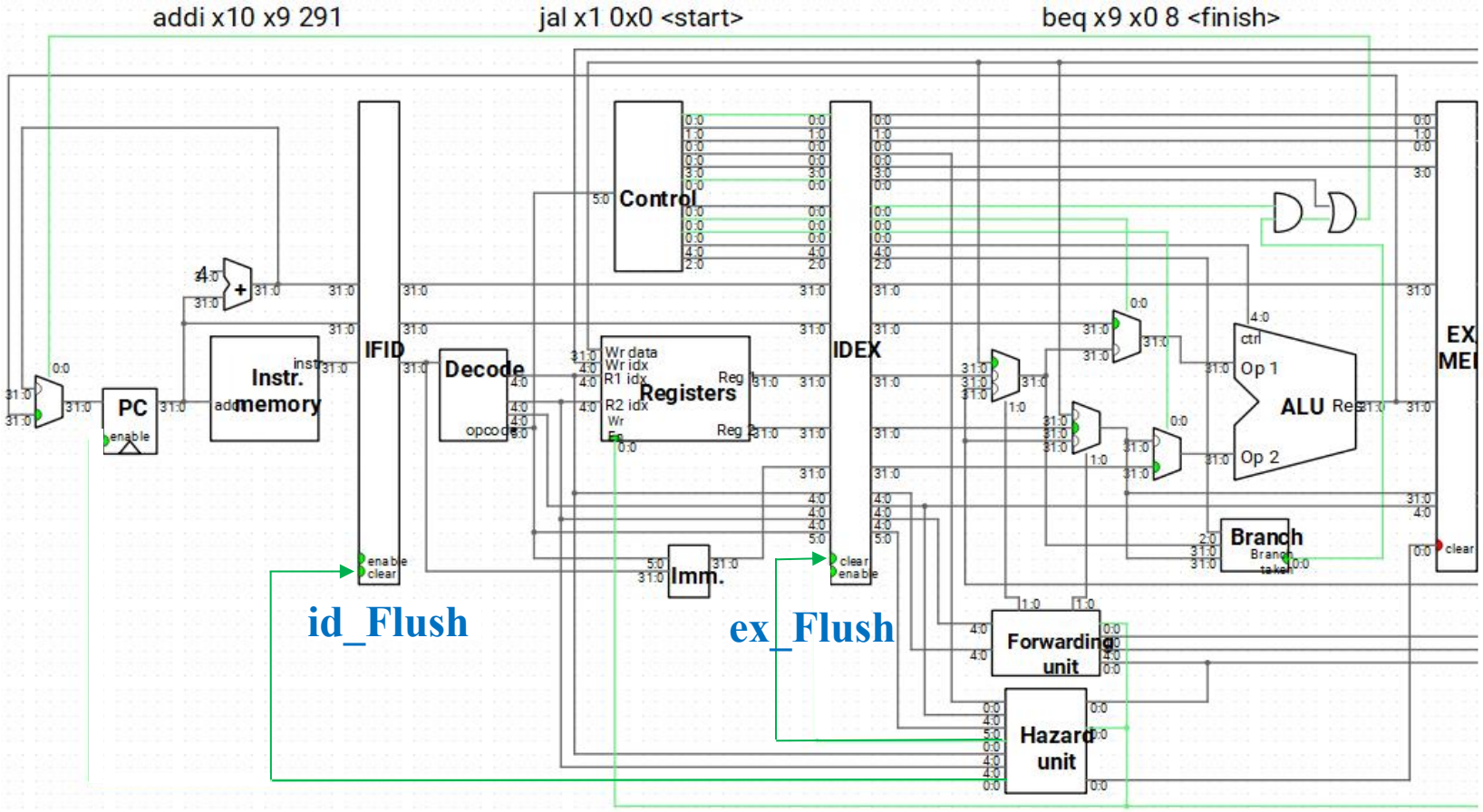
(ID/EX.clr=0) and (IF/ID.en=0)and (PC.en=0)

d)最后将段间寄存器MEM_WB中从数据存储器取的值通过forwarding_unit前递至下一条指令的ex段

```
module Harzard_detection_Unit(
input [4:0] ID_EX_rdest,
input [4:0] IF_ID_rs1,
input [4:0] IF_ID_rs2,
input  ID_EX_Memread,          //lw指令在EX段，下一条指令译码得到rs1,rs2
output reg control_clr,        //ID_EX寄存器控制信号清0：作为复用器的选择信号，控制选择0
output reg IF_ID_en,           //IF_ID寄存器写使能
output reg PC_en               //PC寄存器写使能
    );

    always @(*)
    begin
    if (((ID_EX_rdest == IF_ID_rs1)|(ID_EX_rdest == IF_ID_rs2)) && ID_EX_Memread)
        begin
            control_clr = 1'b0;
            IF_ID_en = 1'b0;
            PC_en = 1'b0;
        end
     else begin
            control_clr = 1'b1;
            IF_ID_en = 1'b1;
            PC_en = 1'b1;
           end
    end
endmodule
```

实验原理

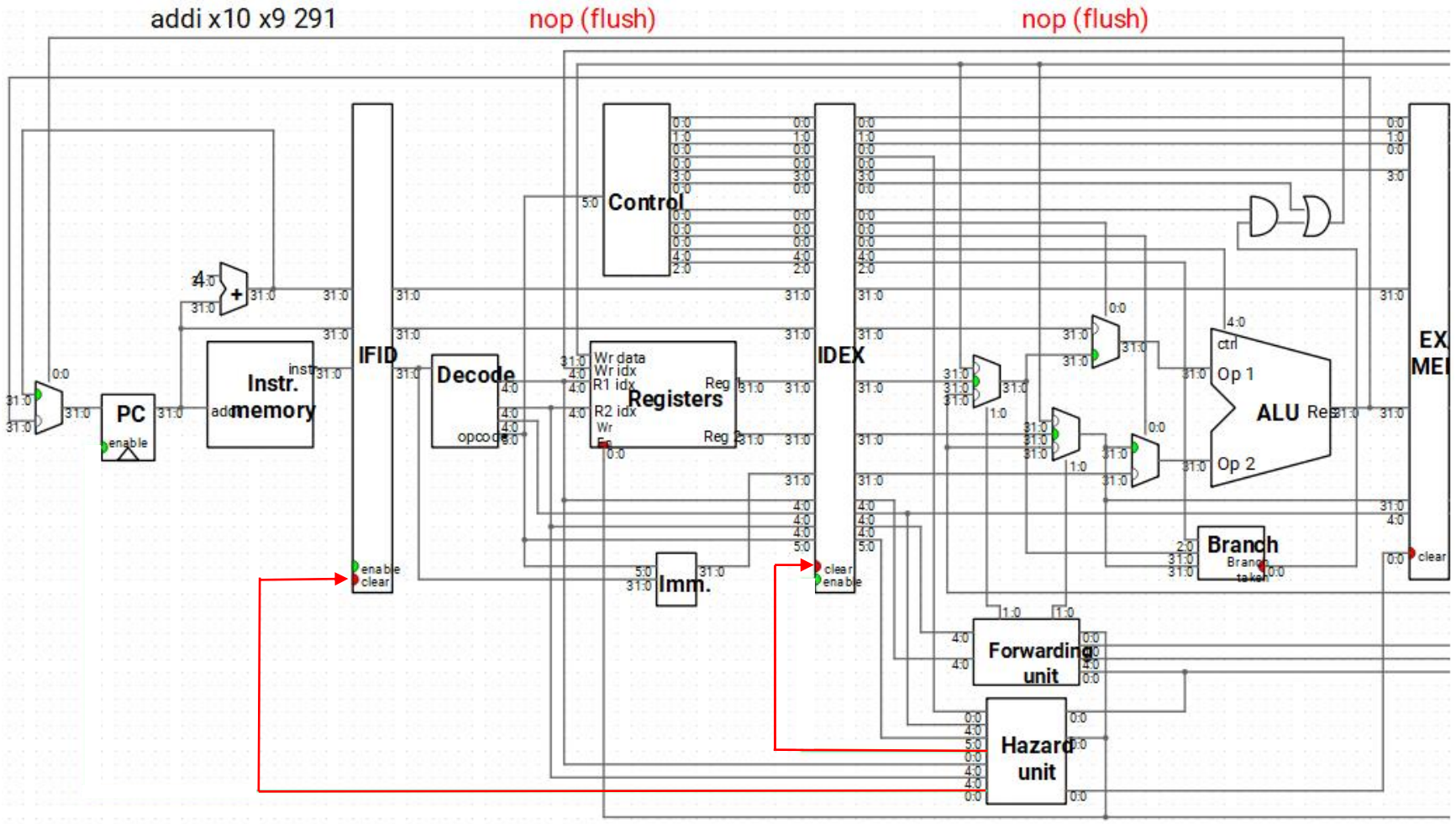
3. 流水线相关及其处理_Branch Hazard (Ripes)



<注意段间寄存器设计时采用同步清空flush>

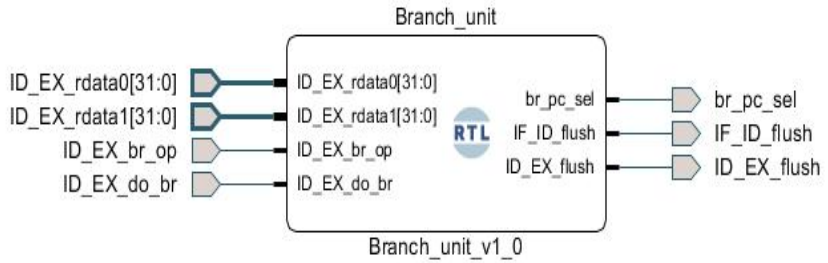
实验原理

3. 流水线相关及其处理_Branch Hazard (Ripes)



实验原理

3. 流水线相关及其处理_Branch Hazard (参考设计)




```
module Branch_unit(
    input [31:0] ID_EX_rdata0,
    input [31:0] ID_EX_rdata1,
    input ID_EX_br_op,ID_EX_do_br,                  //beq
    output br_pc_sel,IF_ID_flush,ID_EX_flush
    );

    wire br_taken;

    assign br_taken = (ID_EX_br_op && (ID_EX_rdata0 == ID_EX_rdata1)) ? 1'b1 : 1'b0;
    assign br_pc_sel = br_taken && ID_EX_do_br;
    assign IF_ID_flush =  br_taken && ID_EX_do_br;
    assign ID_EX_flush =  br_taken && ID_EX_do_br;

endmodule
```

1)检测条件：
当前指令是beq指令，且处于EX段:ID/EX.Branch==1；
分支转移成功：ID/EX.regRd0==ID/EX.regRd1

2)解决方案：假定分支不发生;如果分支发生，就丢弃已经读取并译码的指令，并按分支目标继续执行
a) br_pc_sel= 1 ;（按分支目标继续执行）
b)IF_ID和ID_EX段间寄存器时钟同步输出置0（丢弃已经读取并译码的指令，即清除flush ID级和EX级）
IF_ID_flush = 1;ID_EX_flush = 1;
备注：本数据通路跟课本有所不同，将分支判断放在了EX级，这样只需要IF_ID和ID_EX段间寄存器输出置0，如果放在MEM级，就需要清除IF_ID，ID_EX和EX_MEM段间寄存器输出置0

```
module IF_ID_reg(
    input clk,
    input rst,
    input IF_ID_stall,        //IF_ID_stall为1时，暂停更新段间寄存器
    input IF_ID_flush,
    input [31:0] PC,
    input [31:0] PC_add_4,
    input [31:0] Instruction,
    output reg [31:0] IF_ID_PC,
    output reg [31:0] IF_ID_PC_add_4,
    output reg [31:0] IF_ID_Instruction
    );

    always @(posedge clk, posedge rst)
    begin
        if (rst) begin
            IF_ID_PC              <= 32'd0;
            IF_ID_PC_add_4        <= 32'd0;
            IF_ID_Instruction     <= 32'd0;
        end
        else if (~IF_ID_stall) begin
            IF_ID_PC              <= PC;
            IF_ID_PC_add_4        <= PC_add_4;
            IF_ID_Instruction     <= Instruction;
            end
        else if (IF_ID_flush && (~IF_ID_stall))
            IF_ID_Instruction     <= 32'd0;
    end
endmodule
```

实验原理

3. 流水线相关及其处理_Branch Hazard (参考设计)

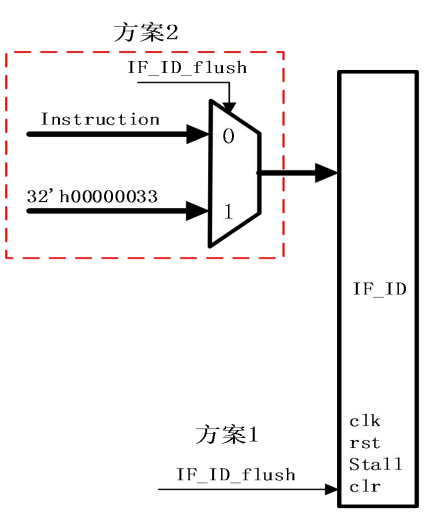
□ Flush的两种实现方案

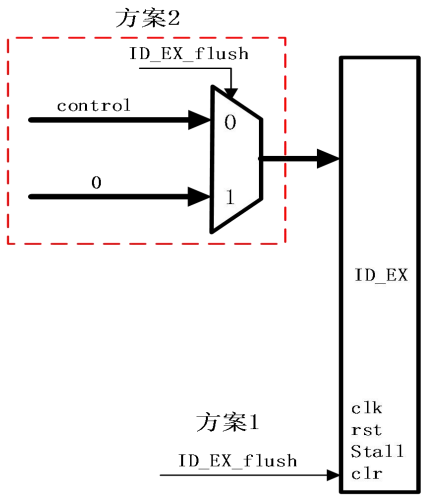
✓ ID级清除

- flush信号作为段间寄存器IF_ID控制信号，将段间寄存器输出同步清0
- flush信号作为复用器选择信号，控制段间寄存器IF_ID输入nop指令（32'h00000033）

✓ EX级清除

- flush信号作为段间寄存器ID_EX控制信号，将段间寄存器输出同步清0
- flush信号作为复用器选择信号，控制段间寄存器ID_EX输入控制信号全0





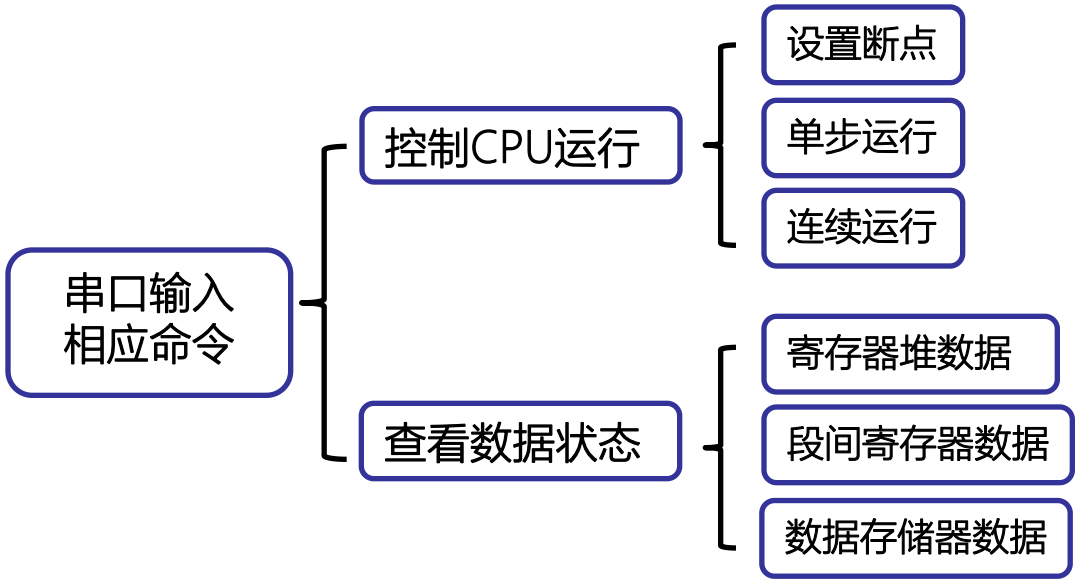
注意flush清0前需要判断stall是否有效

实验原理

4.PDU_PL:外设管理与调试单元----提供PDU_PL代码及说明手册

□ PDU_PL 功能

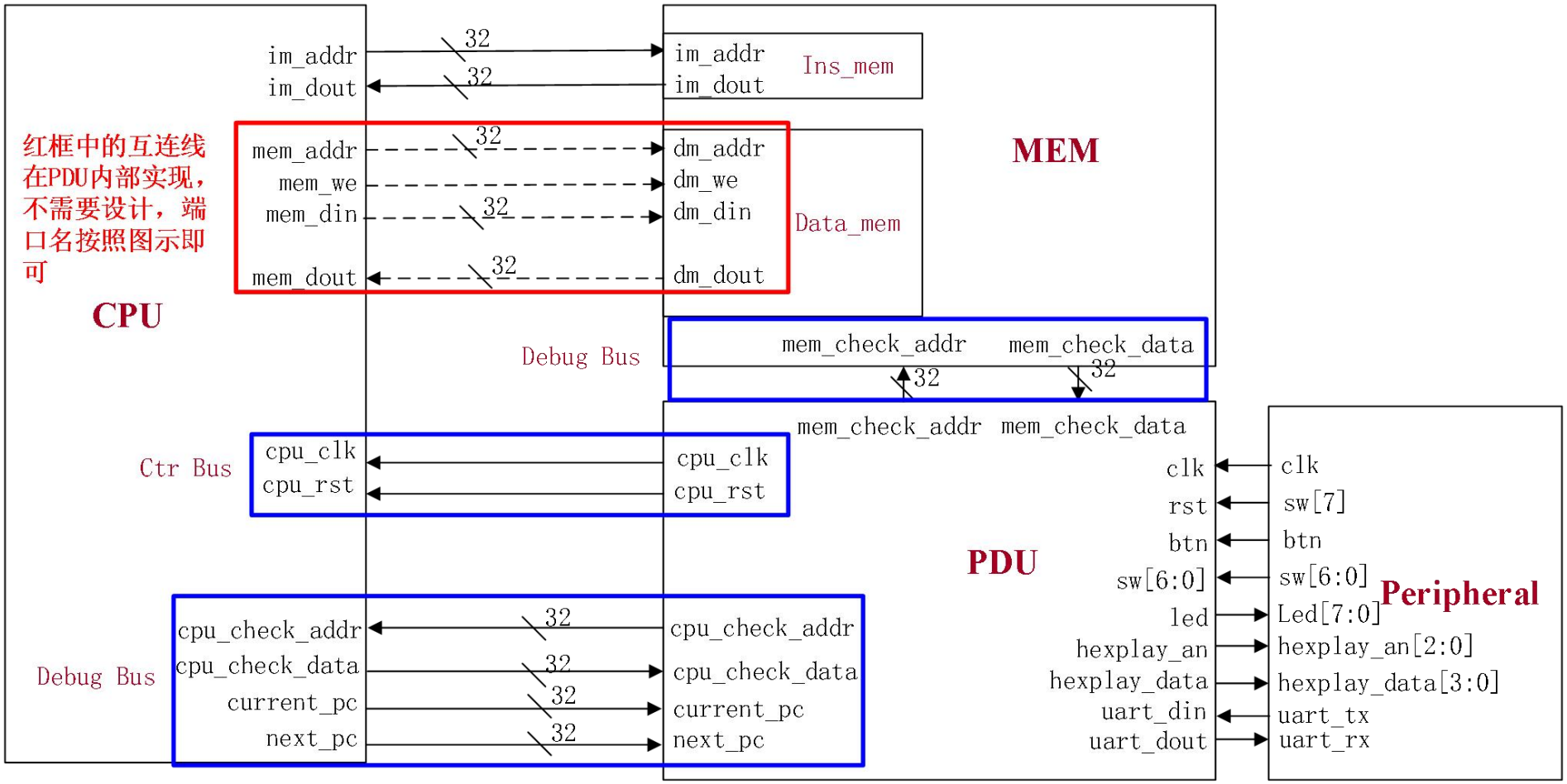
- ✓ **通过串口控制流水线CPU运行方式，及查看流水线CPU状态寄存器数据和数据路径上的数据，详见《PDU_PL指令手册》**



实验原理

4.PDU_PL:外设管理与调试单元----提供PDU_PL代码及说明手册

□ 系统框图

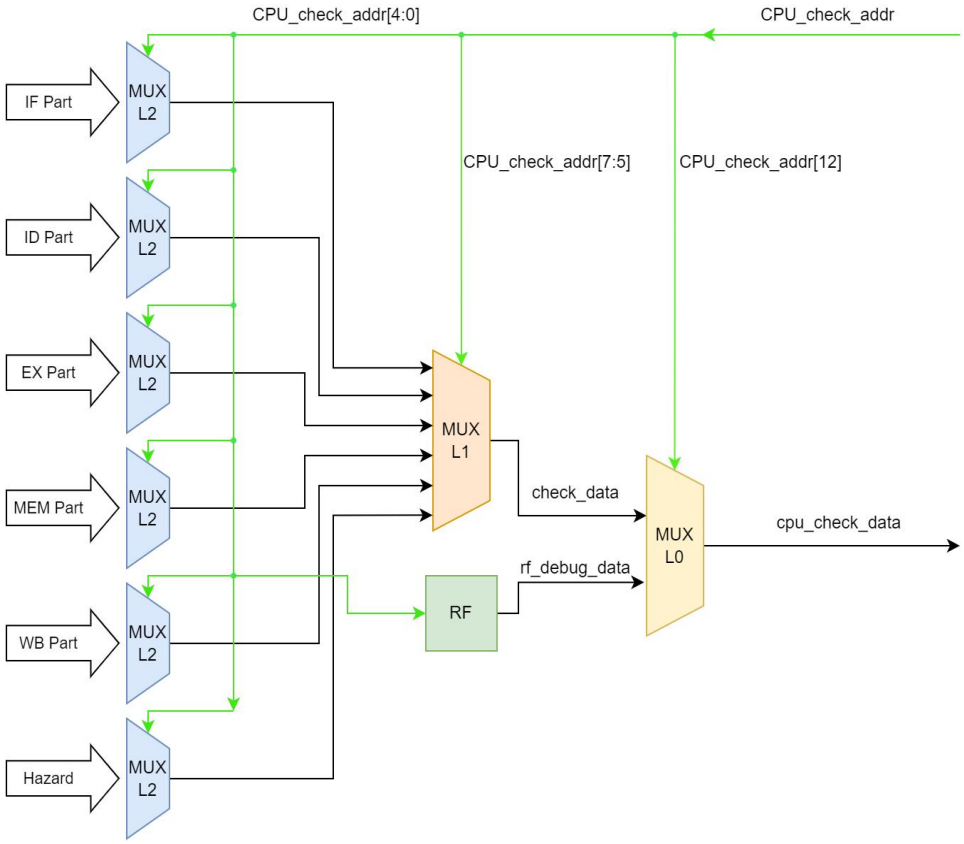


实验原理

4.PDU_PL:外设管理与调试单元----提供PDU_PL代码及说明手册

- 流水线 PDU_PL 的基本操作同单周期 PDU
- debug接口的调整：
  - ✓ 在CPU暂停时，可以依次查看CPU段间寄存器数据、寄存器堆数据及数据存储器数据；
  - ✓ 跟CPU运行程序无关;
  - ✓ cpu_check_addr[7:5]选择不同段间寄存器;
  - ✓ cpu_check_addr[4:0]选择段间寄存器内不同信号;
  - ✓ cpu_check_addr[12]选择段间寄存器或寄存器堆。

<具体信号选择参考PDU指令手册>



实验原理

4.PDU_PL:外设管理与调试单元----提供PDU_PL代码及说明手册

□ 系统工作状态指示

数值 / Led编号	1	0
led[7]	CPU 正在运行	CPU暂停
led[6]	数码管显示开关输入数据	其他
led[5]	数码管显示 CPU 输出数据	其他
led[4]	数码管显示 DEBUG 数据	其他
led[1]	指令检测失败	其他
led[0]	指令检测成功	其他

实验要求[必做]

1. 设计五级流水线CPU并进行功能仿真

- **CPU数据通路需支持以下十条指令：**
  - ✓ add addi lui auipc lw sw beq blt jal jalr
- **修改Lab4寄存器堆模块，使其满足写优先(Write First)**
  - ✓ 即在对同一寄存器读写时（0号寄存器除外），将要写的数据从读数据端口输出
- **存储器参数设置（同单周期CPU)**
  - ✓ 指令存储器采用分布式ROM(256x32位)，地址范围：0x0000_3000 ~ 0x0000_33ff
  - ✓ 数据存储器采用分布式Dual Port Ram(256x32位)，地址范围：0x0000_0000 ~ 0x0000_03ff，其中一个读端口用于调试
- **结构化描述流水线CPU**
  - ✓ vivado工程结构：

```
|--+top.v
|   |--MEM.v:内部例化数据存储器和指令存储器IP （需要例化IP）
|   |--CPU.v:流水线数据通路 （需要设计）
|   |--PDU.v:外设管理及调试单元 （不需要设计，可提供所有代码）
```

- **对流水线CPU进行功能仿真**

实验要求[必做]

2. 对流水线CPU进行下载测试

- **测试无数据和控制相关的汇编程序**
  - ✓ 将CPU和PDU连接，加载simple_test.asm（自测）；
- **测试有数据相关处理的汇编程序**
  - ✓ 将CPU和PDU连接，加载data_test.asm（自测）；
- **测试有控制相关处理的汇编程序**
  - ✓ 将CPU和PDU连接，加载control_test.asm（自测）；
- **测试同时有数据相关和控制相关的汇编程序**
  - ✓ 将CPU和PDU连接，加载pipeline_test.asm（最终检查）。

实验要求[选做]

1. 缩短jal指令延迟

- 将jal目标地址计算提前至ID阶段;
- 设计提示:
  - ✓ 可通过control_test.asm进行自测

2. ebreak指令的实现

- 仅实现程序结束的功能即可;
- 可预留恢复接口供综合实验使用;
- 通过ebreak_test.asm进行自测。

The End