Computer Organization & Design实验与课程设计

Lab05-1
CPU设计—流水线处理器集成

Ma De（马德）

made@zju.edu.cn

2025

College of Computer Science, Zhejiang University

Course Outline

- 一、实验目的
- 二、实验环境
- 三、实验目标及任务

实验目的

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

实验环境

- 实验设备

  1. 计算机（Intel Core i5以上，4GB内存以上）系统

  2. NEXYS A7开发板

  3. VIVADO 2017.4及以上开发工具

- 材料

  无

实验目标及任务

- 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

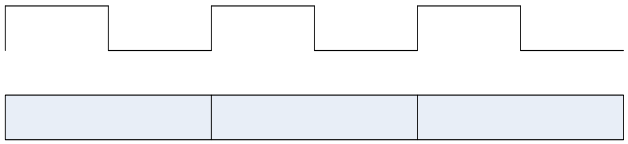
- 任务一：集成设计流水线CPU，在Exp04的基础上完成
  - 利用五级流水线各级封装模块集成CPU
  - 替换 Exp04的单周期CPU为本实验集成的五级流水线CPU

- 任务二：设计流水线测试方案并完成测试

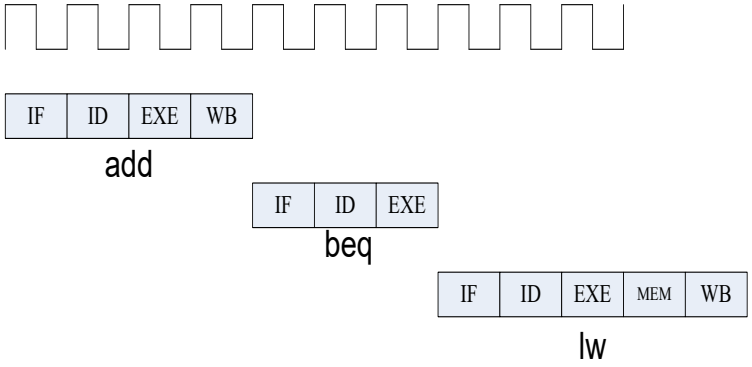
RISC-V 流水线处理器的原理介绍

Why pipeline

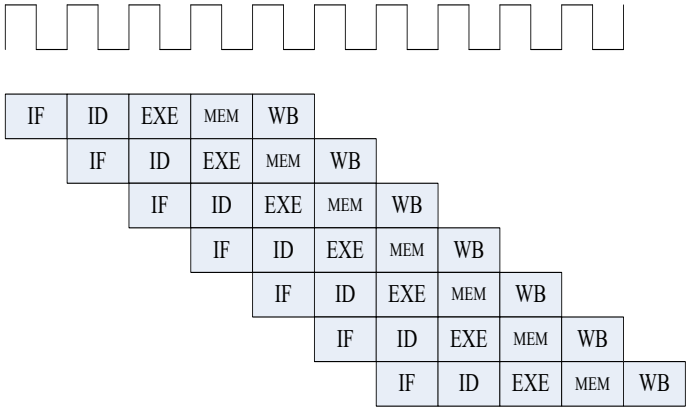
Simple-Cycle CPU



Multiple-Cycle CPU



Pipelined CPU



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

-------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

-------效率提高

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

-------效率更高

Pipelining with RISC-V

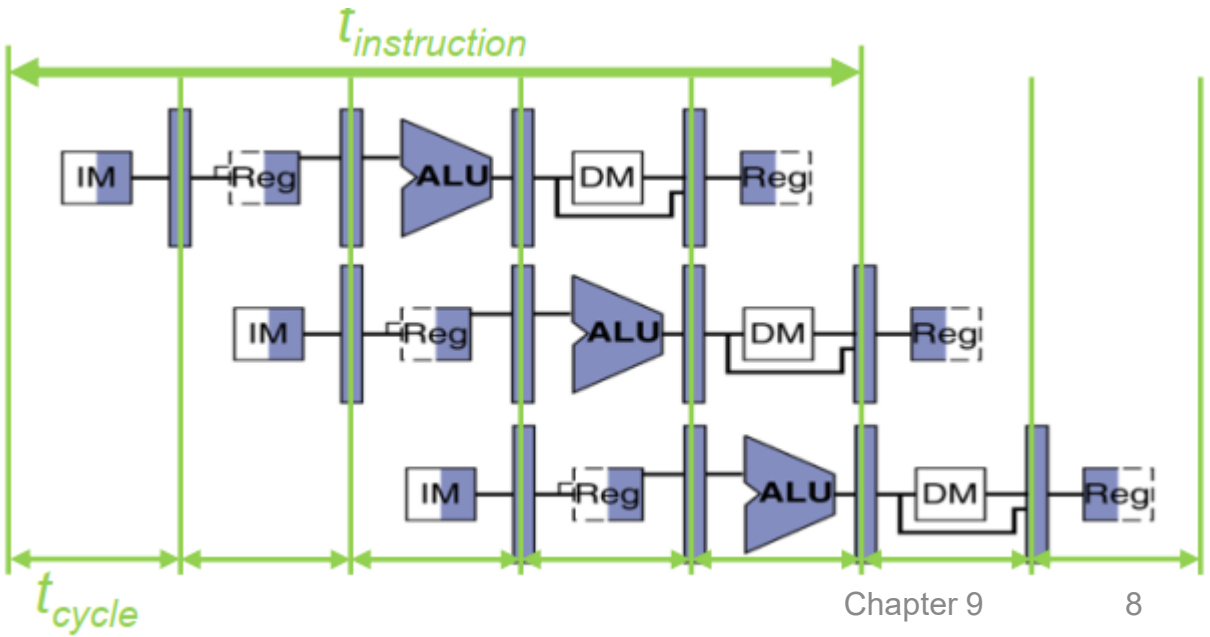
Phase	Pictogram	t_{step} Serial	t_{cycle} Pipelined
Instruction Fetch		200 ps	200 ps
Reg Read		100 ps	200 ps
ALU		200 ps	200 ps
Memory		200 ps	200 ps
Register Write		100 ps	200 ps
$t_{instruction}$		**800 ps**	**1000 ps**

instruction sequence

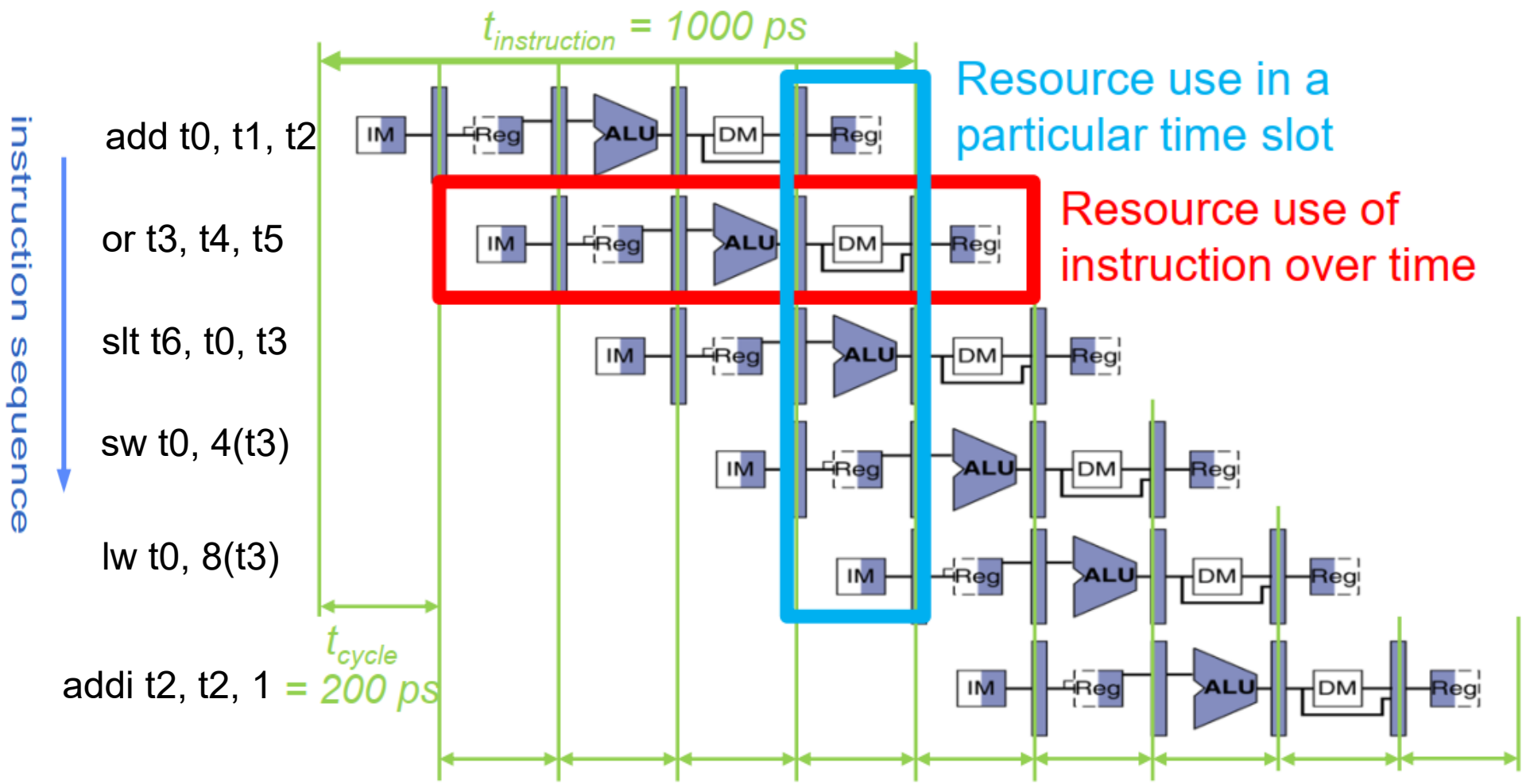
add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3



Pipelining with RISC-V



Pipelining with RISC-V

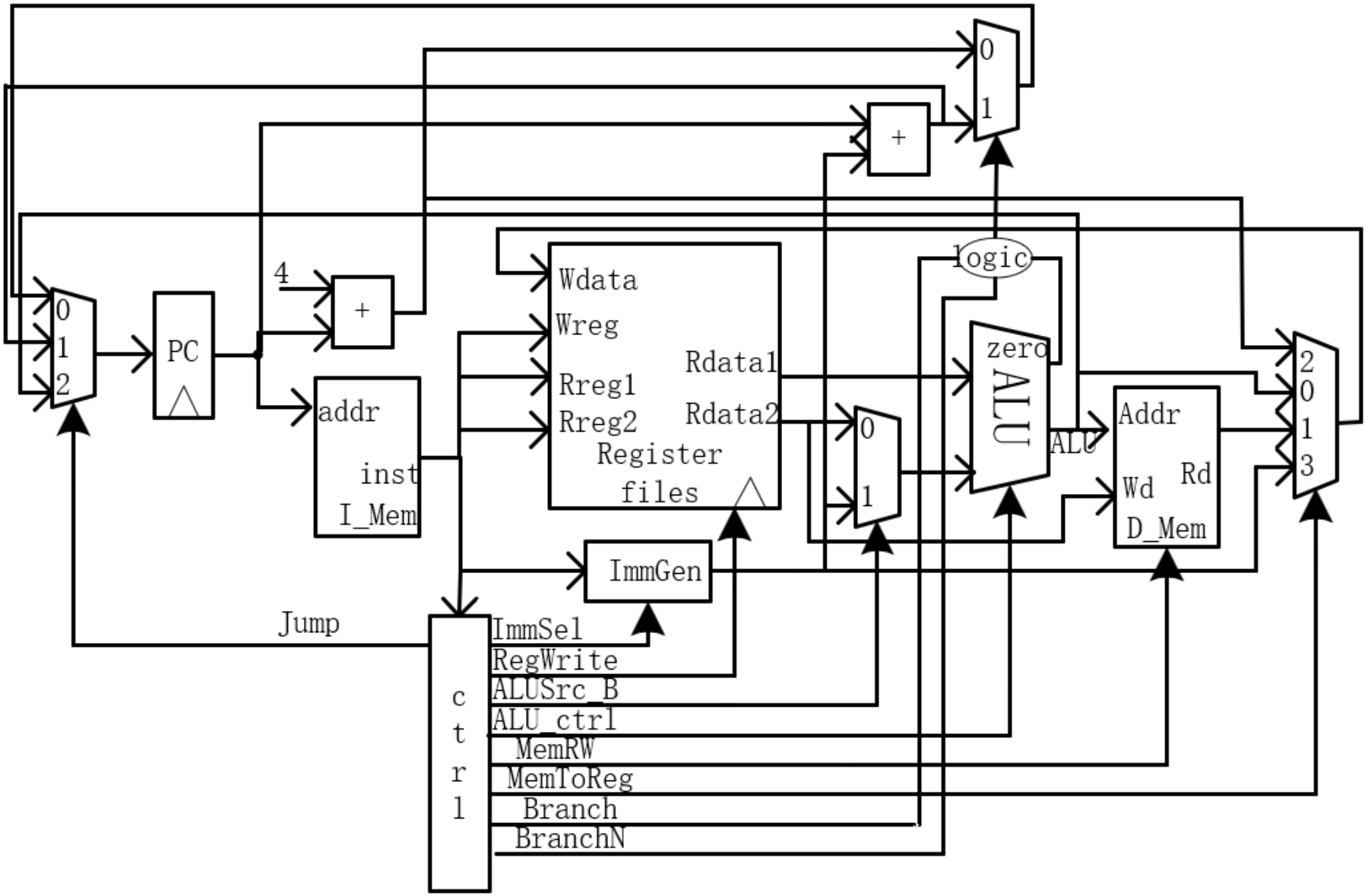
五级流水线的每一级的具体操作

指令 / 流水段	ALU	LOAD/STORE	BRANCH
IF	取指令	取指令	取指令
ID	译码，读寄存器文件	译码，读寄存器文件	译码，读寄存器文件
EXE	执行	计算访存有效地址	计算转移目标地址，设置条件码
MEM	(空操作)	对存储器读或写操作	若条件成立，将转移地址送 PC
WB	结果写入寄存器文件	读出数据写入寄存器文件	(空操作)

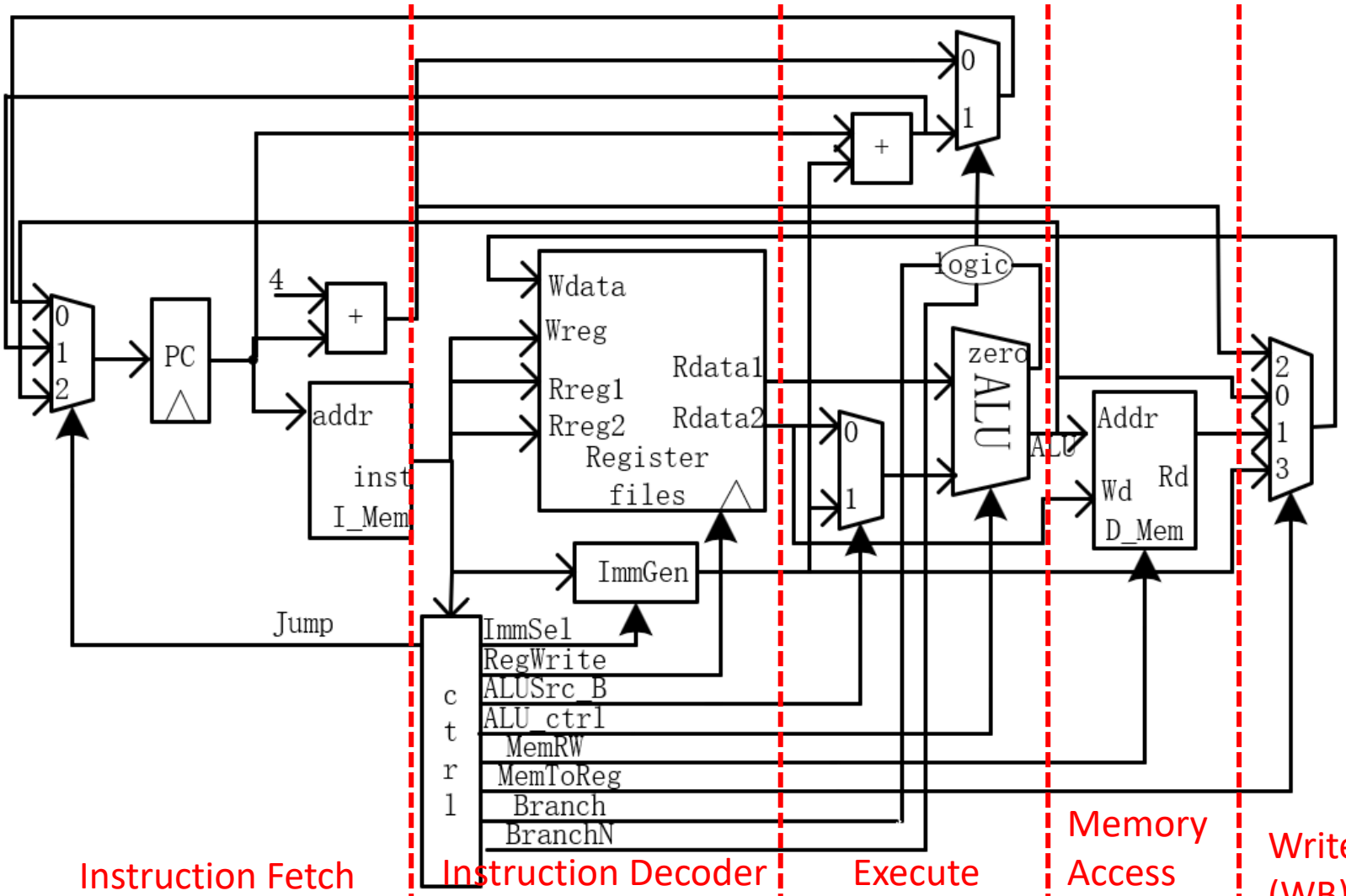
指令流水线的时空图

指令序列	流水时钟数								
	1	2	3	4	5	6	7	8	9
指令 i	IF	ID	EX	MEM	WB				
指令 i +1		IF	ID	EX	MEM	WB			
指令 i +2			IF	ID	EX	MEM	WB		
指令 i +3				IF	ID	EX	MEM	WB	
指令 i +4					IF	ID	EX	MEM	WB

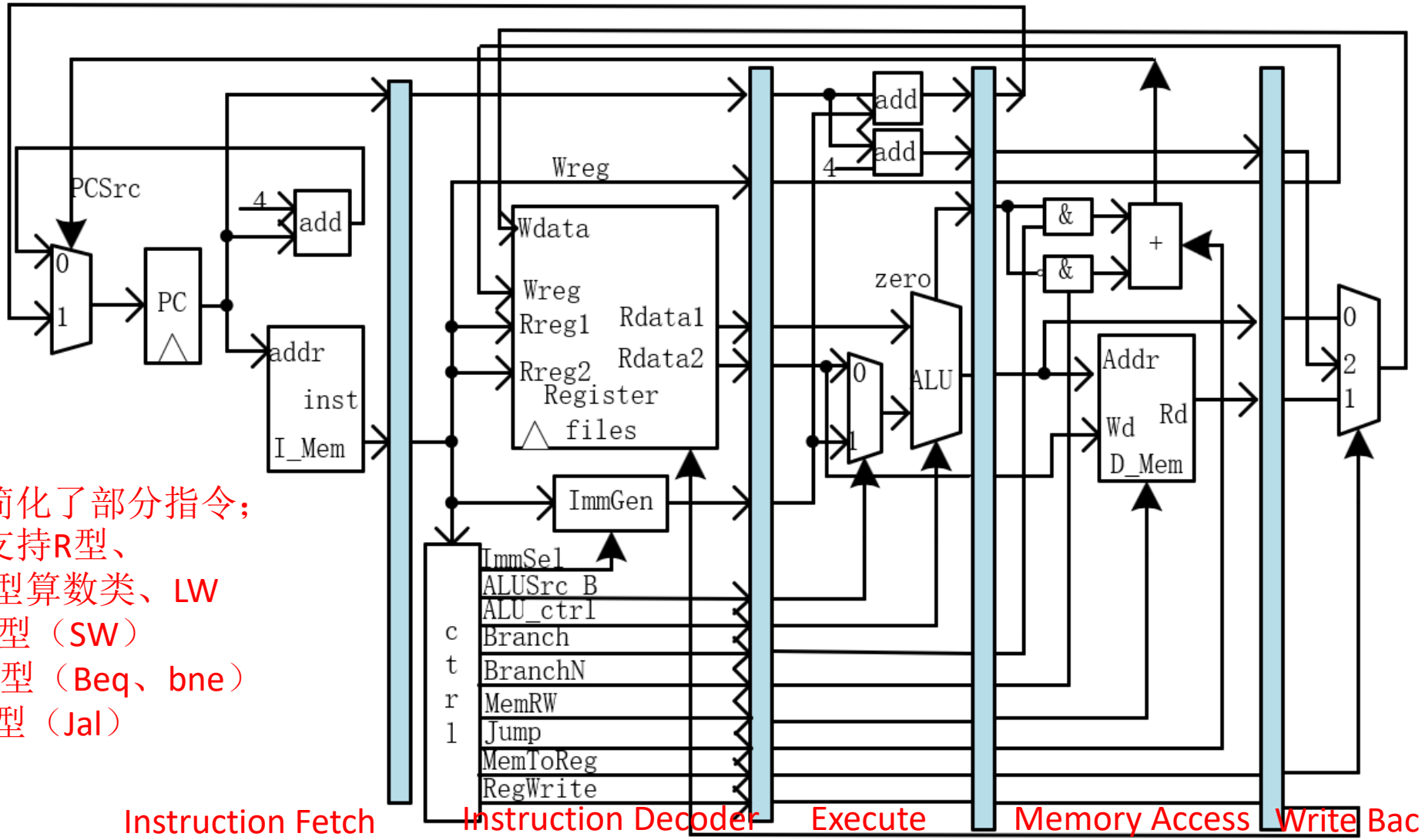
Single-Cycle RISC-V RV32I Datapath



Pipelining RISC-V RV32I Datapath



Pipelined RISC-V RV32I Datapath



简化了部分指令；
支持R型、
I型算数类、LW
S型（SW）
B型（Beq、bne）
J型（Jal）

Instruction Fetch (IF)

Instruction Decoder (ID)

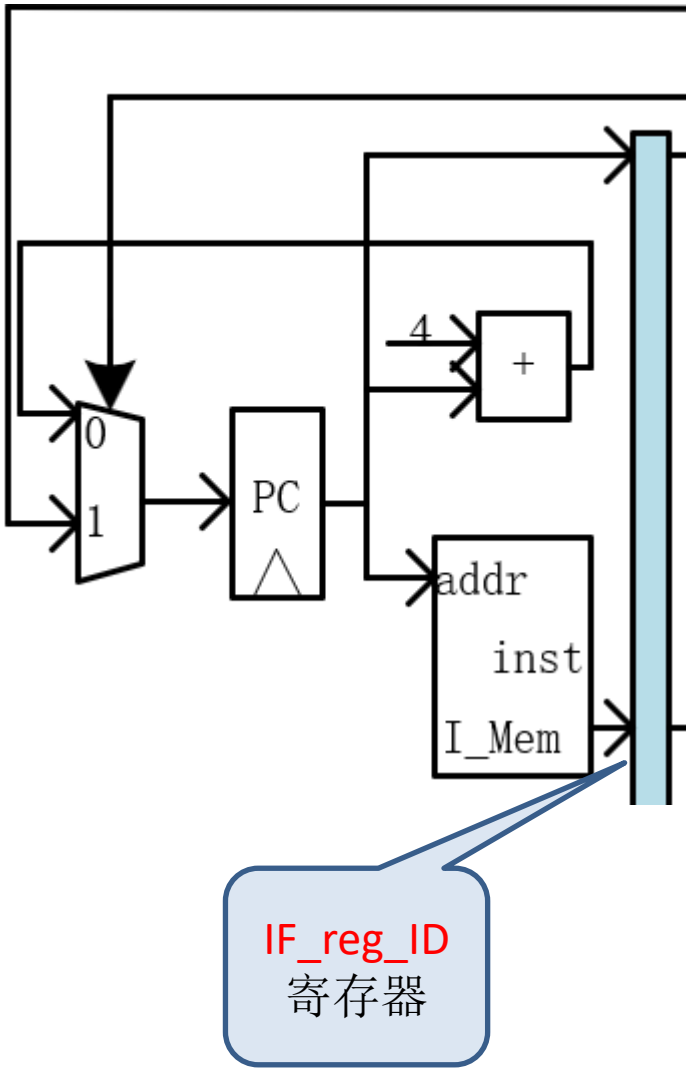
Execute (Ex)

Memory Access (Mem)

Write Back (WB)

Pipeline

Instruction Fetch (IF)

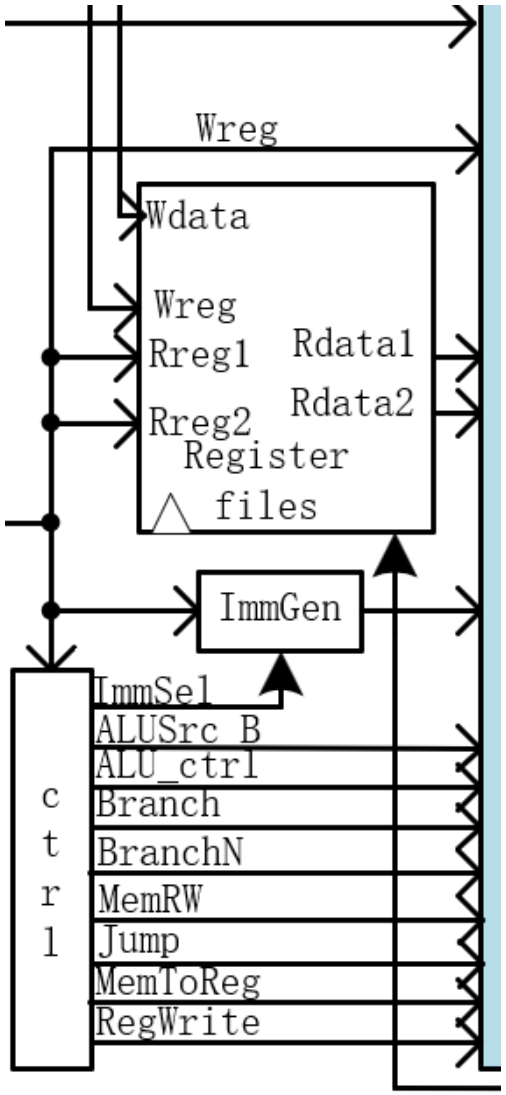


- **取指**：取指阶段涉及程序计数器（PC）和指令存储器（I_Mem）；程序计数器输出作为地址从指令存储器中读取指令。
- **IF_reg_ID**：暂存指令和PC值，以待下一级使用

通过引入寄存器保存数据的方式，使得部分数据通路可以在执行指令的过程中被共享，所以需要插入四级寄存器切分五级流水。

Pipeline

Instruction Decoder(ID)



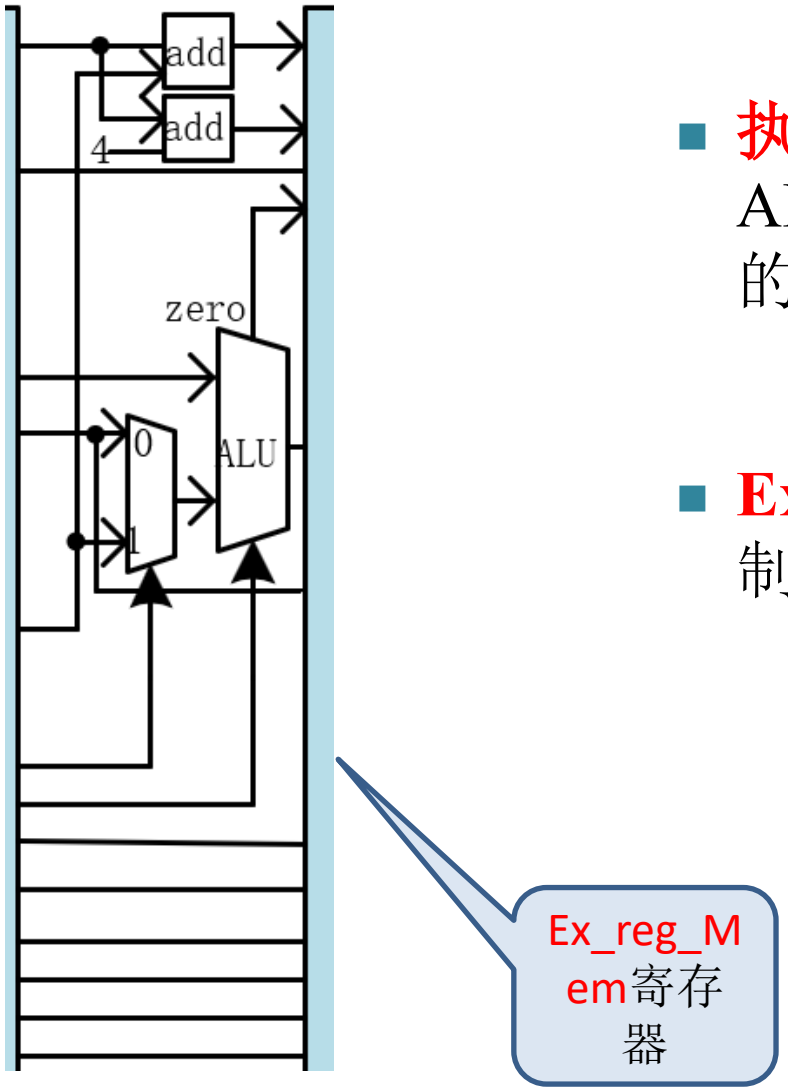
译码器即为单周期CPU中的控制器

- **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

ID_reg_Ex 寄存器

Pipeline

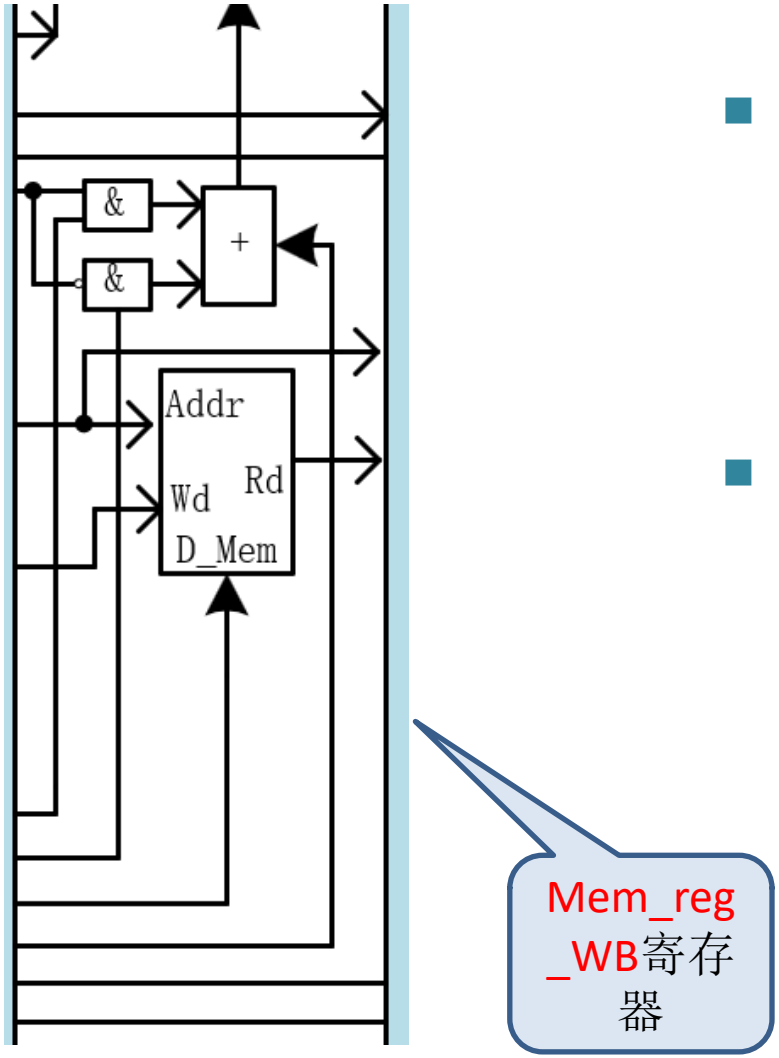
Execute(Ex)



- **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

Pipeline

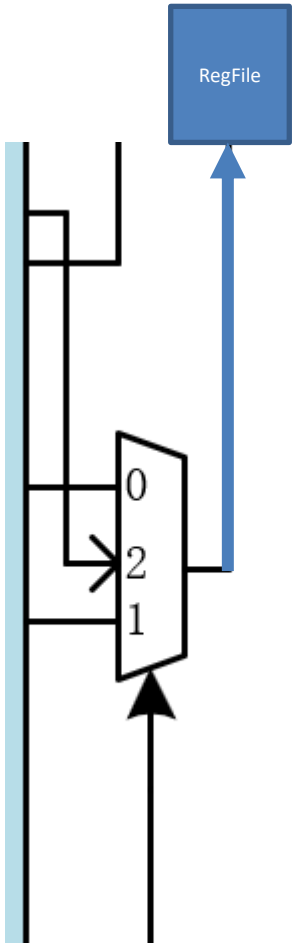
Memory Access(Mem)



- **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

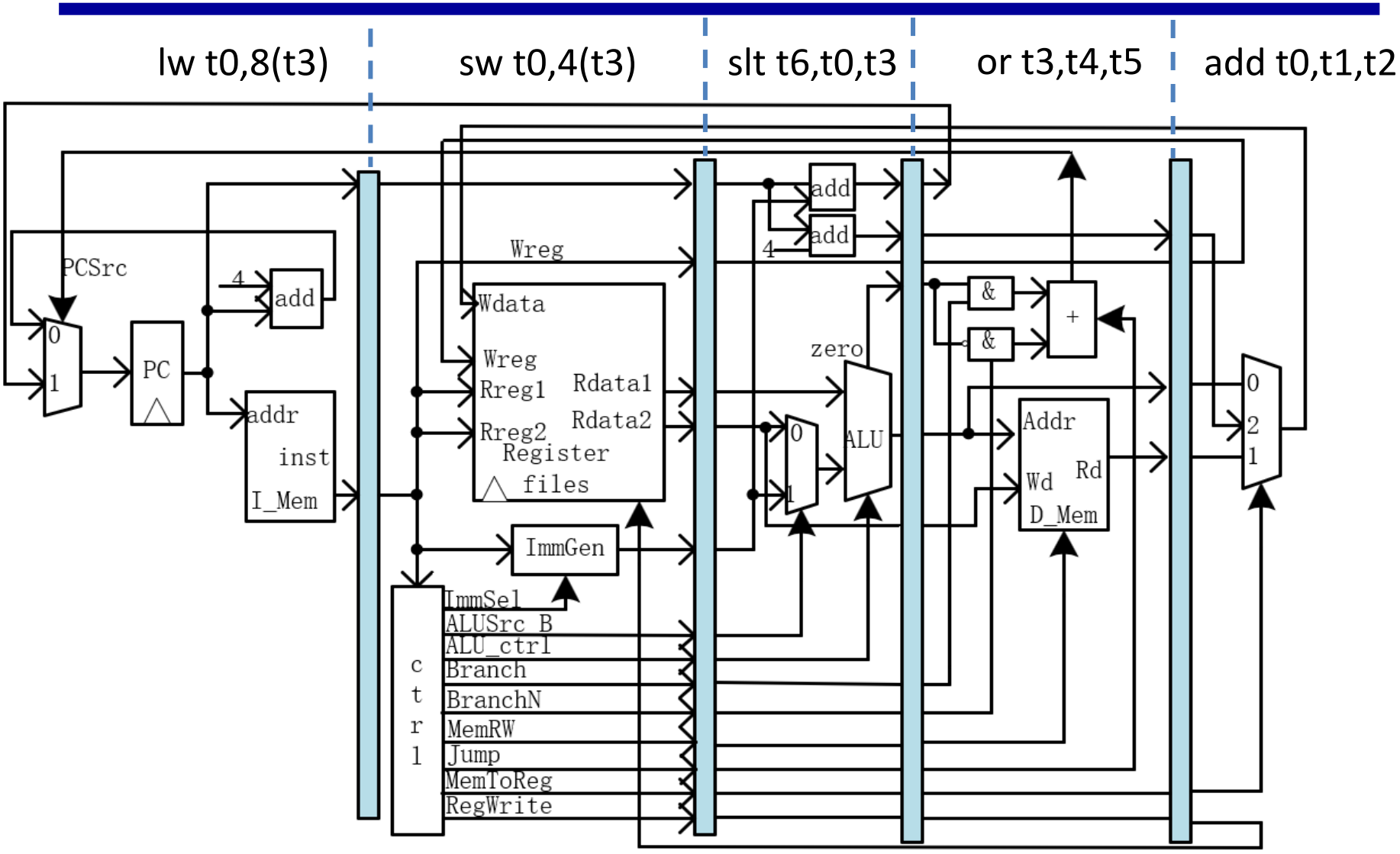
Pipeline

Write Back(WB)



- **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。

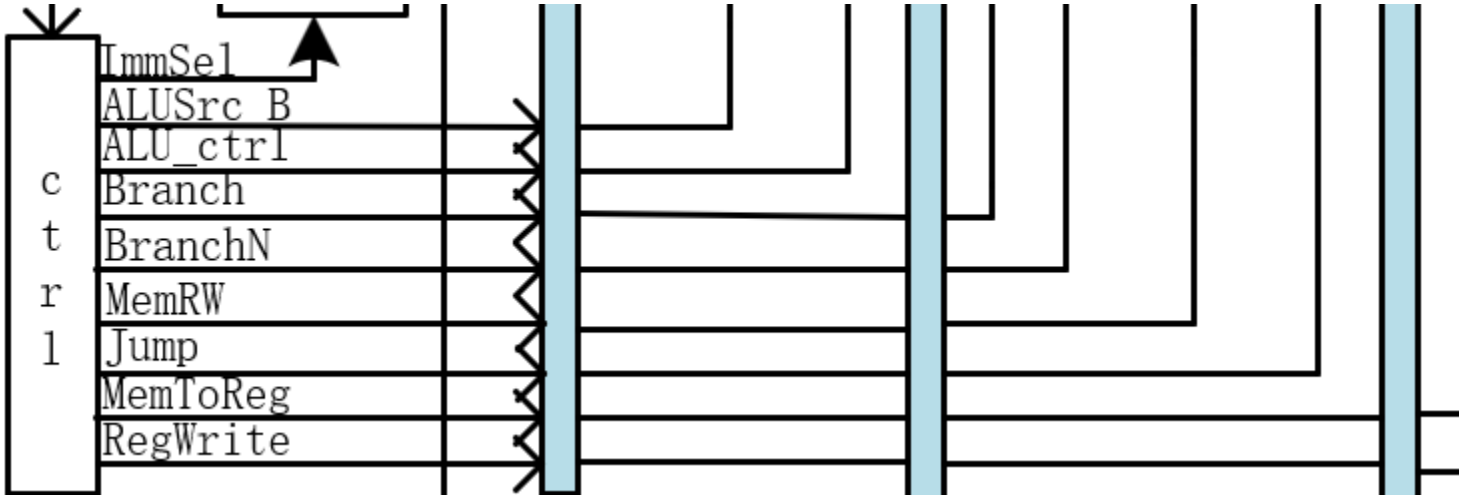
Each stage operates on different instruction



Pipeline registers separate stages, hold data for each instruction in flight

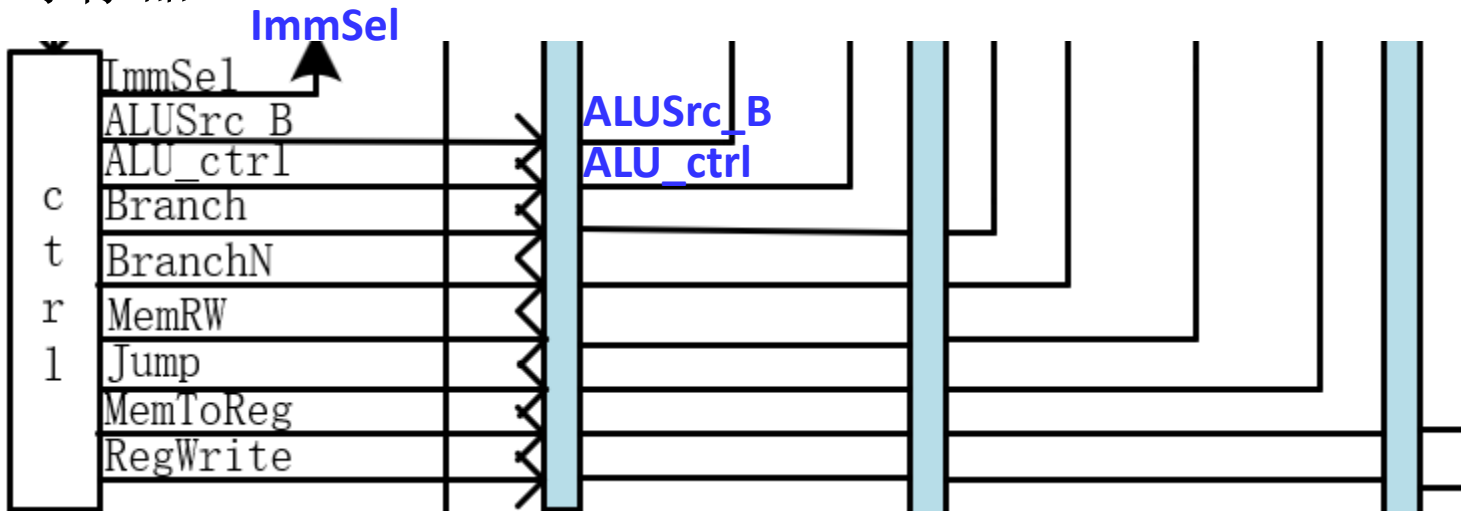
Pipelined RISC-V RV32I Control

- **流水线控制**：流水线的控制信号同单周期CPU一样，来自于对指令的译码操作输出；不同点在于流水控制信号会根据每阶段的不同功能部件选择性控制输出，本阶段用不到的信号会暂存于寄存器。



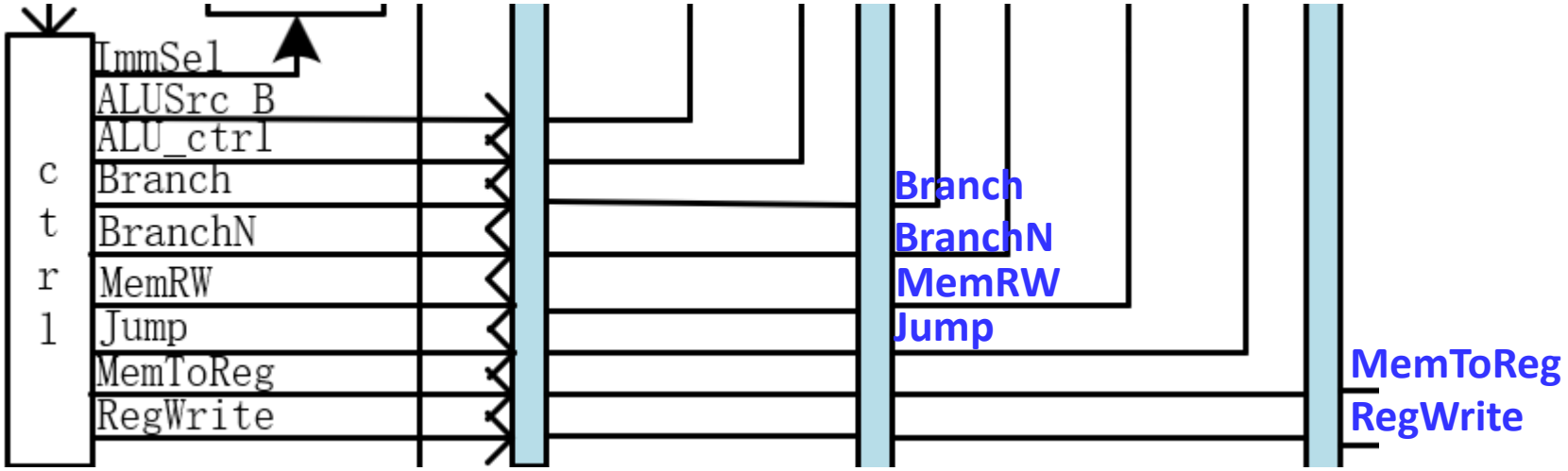
Pipelined RISC-V RV32I Control

- **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。



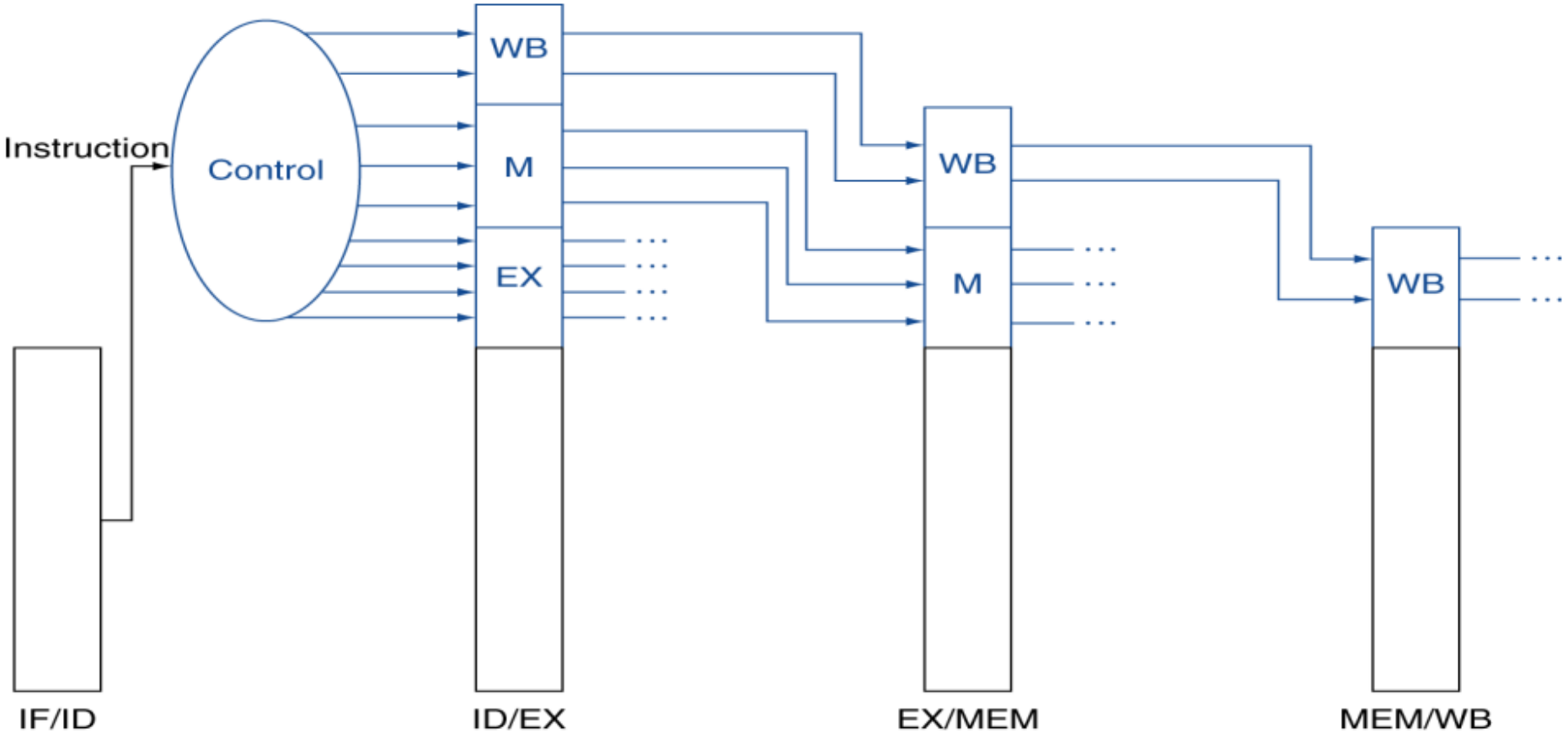
Pipelined RISC-V RV32I Control

- **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。



Pipelined RISC-V RV32I Control

- 控制信号的产生来源于指令
  - 控制器即译码单元的实现与单周期的实现相同
  - 控制信号生成后备暂存于pipelined register以备后续使用



Pipelined RISC-V RV32I Control

信号	源数目	功能定义	赋值0时动作	赋值1时动作	赋值2时动作
ALUSrc_B	2	ALU端口B输入选择	选择源操作数寄存器2数据	选择32位立即数（符号扩展后）	-
MemToReg	3	寄存器写入数据选择	选择ALU输出	选择存储器数据	选择PC+4
Branch		Beq指令目标地址选择	选择PC+4地址	选择转移目的地址 PC+imm（zero=1）	-
BranchN		Bne指令目标地址选择	选择PC+4地址	选择转移目的地址 PC+imm（zero=0）	-
Jump		Jal指令目标地址选择	选择PC+4地址	选择跳转目的地址	-
PCSrc	2	PC输入选择（分支跳转的衍生）	=(Branch&zero)|(BranchN&(~zero))|Jump		-
RegWrite	-	寄存器写控制	禁止寄存器写	使能寄存器写	-
MemRW	-	存储器读写控制	存储器**读使能**，存储器写禁止	存储器**写使能**，存储器读禁止	-
ALU_Control	000-111	3位ALU操作控制	参考表ALU_Control（详见实验04）		
ImmSel	00-11	2位立即数组合控制	参考表ImmSel（详见实验04）		

Pipelining Hazards

- 以上内容均是在五级流水线正常执行的情况，然而CPU实际运行时，会发生下一条指令无法在下个时钟周期正常执行的情况，即为流水线冒险。
- 导致流水线冒险的原因有多种，具体在Lab05-4详细介绍
- 本次实验暂不考虑流水线冒险的情况

- **任务一**：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

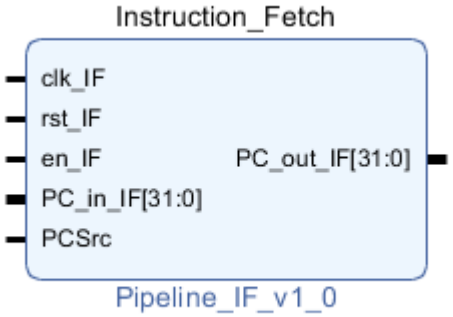
采用代码组织的方式，具体方法请参考lab04

CPU之五级流水线集成

-根据五个阶段流水线模块集成CPU

Pipeline—IF　　取指模块接口

- **Pipeline_IF**
  - 流水线CPU第一阶段
  - 根据程序计数器从指令存储器中取出指令
- **基本功能**
  - 由程序计数器获取PC值及PC值的更新
  - 由PC值获取指令
- **接口要求**
  - 取指模块接口如图：



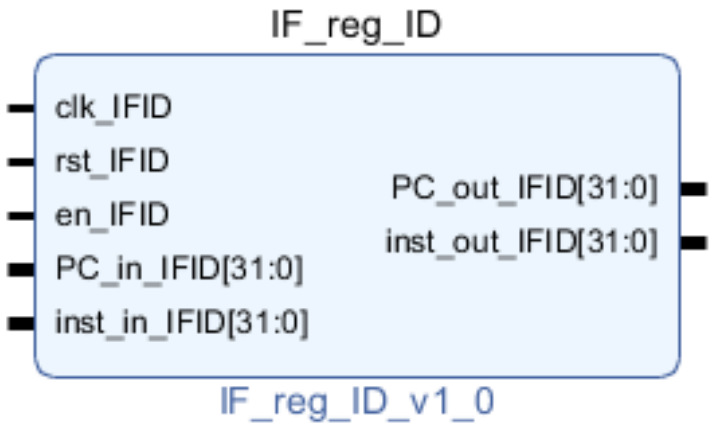
取指模块接口信号标准：Pipeline_IF.v

```
module     Pipeline_IF(
             input                clk_IF,        //时钟
             input                rst_IF,        //复位
             input                en_IF,         //使能
            input        [31:0]   PC_in_IF,     //取指令PC输入
             input                PCSrc,         //PC输入选择
             output reg [31:0]    PC_out_IF     //PC输出
                          );
endmodule
```

（PCSrc=(Branch&zero)|(BranchN&(~zero))|Jump）

Pipeline—IF_reg_ID　　取指-译码寄存器接口

- **IF_reg_ID**
  - 流水线CPU取指和译码之间的寄存器
  - 存储PC值和指令
- **基本功能**
  - 寄存IF级的输出指令，分割IF级和ID级的指令或控制信号，防止相互干扰，在IF级执行结束时将指令的控制信号传递至下一级。
- **接口要求**
  - 取指译码寄存器接口如图：



取指-译码寄存器接口：IF_reg_ID.v

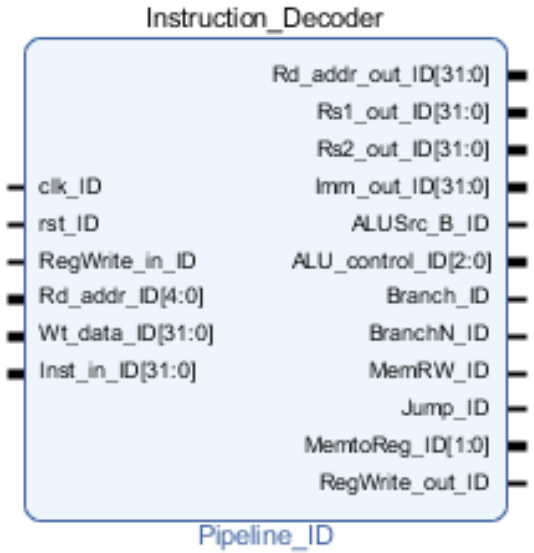
```
module    IF_reg_ID(
            input                clk_IFID,          //寄存器时钟
            input                rst_IFID,          //寄存器复位
            input                en_IFID,           //寄存器使能
            input  [31:0]        PC_in_IFID,        //PC输入
            input  [31:0]        inst_in_IFID,      //指令输入

            output reg [31:0]    PC_out_IFID，      //PC输出
            output reg [31:0]    inst_out_IFID      //指令输出

                        );
endmodule
```

Pipeline—ID　译码模块接口

- **Pipeline_ID**
  - 流水线CPU第二阶段
  - 指令译码
- **基本功能**
  - 译码是指将从指令存储器取指的指令进行翻译的过程；译码之后产生各种控制信号，同时寄存器堆根据所需操作数寄存器的索引读出操作数，立即数生成单元输出所需立即数。
- **接口要求**
  - 译码模块接口如图：



译码模块接口：Pipeline_ID.v

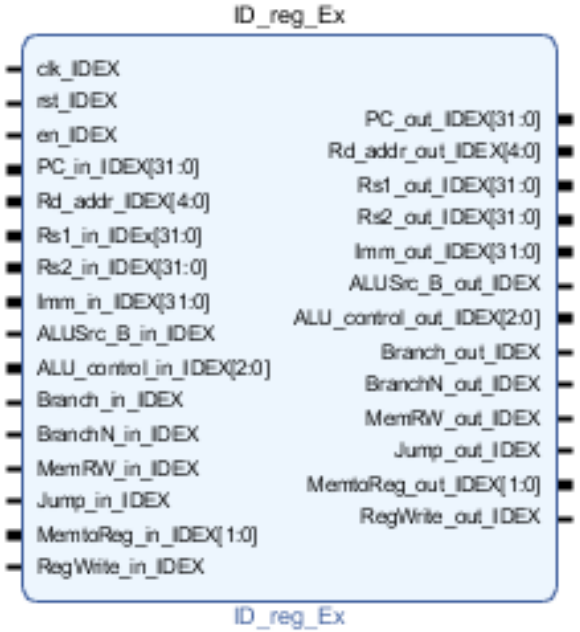
```
module   Pipeline_ID(
          input            clk_ID,              //时钟
          input            rst_ID,              //复位
          input            RegWrite _in_ID,     //寄存器堆使能
          input  [4:0]     Rd_addr_ID,          //写目的地址输入
          input  [31:0]    Wt_data_ID,          //写数据输入
          input  [31:0]    Inst_in_ID,          //指令输入
                 ……
```

译码模块接口：Pipeline_ID.v

```
……
output reg [4:0]   Rd_addr_out_ID,  //写目的地址输出
output reg [31:0]   Rs1_out_ID，      //操作数1输出
output reg [31:0]   Rs2_out_ID ,      //操作数2输出
output reg [31:0]   Imm_out_ID ,      //立即数输出
output reg          ALUSrc_B_ID ,     //ALU B端输入选择
output reg [2:0]    ALU_control_ID,   //ALU控制
output reg          Branch_ID,        //Beq控制
output reg          BranchN_ID,       //Bne控制
output reg          MemRW_ID,         //存储器读写
output reg          Jump_ID,          //Jal控制
output reg [1:0]    MemtoReg_ID,      //寄存器写回选择
output reg          RegWrite_out_ID,  //寄存器堆读写
                );
endmodule
```

Pipeline—ID_reg_Ex　　译码-执行寄存器接口

- **ID_reg_Ex**
  - 流水线CPU译码和执行之间的寄存器
  - 存储ALU数据和控制信号
- **基本功能**
  - 寄存ID级的输出指令，分割ID级和EX级的指令或控制信号，防止相互干扰，在ID级执行结束时将指令的控制信号传递至下一级。。
- **接口要求**
  - 译码执行寄存器接口如图：



译码-执行寄存器接口：ID_reg_Ex.v

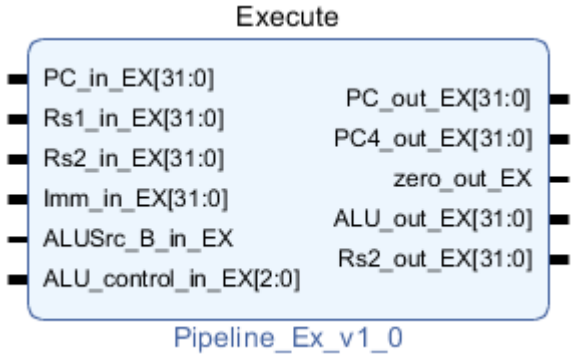
```
module    ID_reg_Ex(
           input          clk_IDEX,                //寄存器时钟
           input          rst_IDEX,                //寄存器复位
           input          en_IDEX,                 //寄存器使能
           input[31:0]    PC_in_IDEX,              //PC输入
           input[4:0]     Rd_addr_IDEX,            //写目的地址输入
           input[31:0]    Rs1_in_IDEX，            //操作数1输入
           input[31:0]    Rs2_in_ID EX,            //操作数2输如
           input[31:0]    Imm_in_IDEX ,            //立即数输入
           input          ALUSrc_B_in_IDEX ,       //ALU B输入选择
           input[2:0]     ALU_control_in_IDEX,     //ALU选择控制
           input          Branch_in_IDEX,          //Beq
           input          BranchN_in_IDEX,         //Bne
           input          MemRW_in_IDEX,           //存储器读写
           input          Jump_in_IDEX,            //Jal
           input[1:0]     MemtoReg_in_IDEX,        //写回选择
           input          RegWrite_in_IDEX,        //寄存器堆读写
           ……
```

译码-执行寄存器接口：ID_reg_Ex.v

```
.....................................
                output reg[31:0]   PC_out_IDEX,          //PC输出
                output reg[4:0]    Rd_addr_out_IDEX      //目的地址输出
                output reg[31:0]   Rs1_out_IDEX,         //操作数1输出
                output reg[31:0]   Rs2_out_ID EX,        //操作数2输出
                output reg[31:0]   Imm_out_IDEX ,        //立即数输出
                output reg         ALUSrc_B_out_IDEX ,   //ALU B选择
                output reg[2:0]    ALU_control_out_IDEX, //ALU控制
                output reg         Branch_out_IDEX,      //Beq
                output reg         BranchN_out_IDEX,     //Bne
                output reg         MemRW_out_IDEX,       //存储器读写
                output reg         Jump_out_IDEX,        //Jal
                output reg [1:0]   MemtoReg_out_IDEX,    //写回
                output reg         RegWrite_out_IDEX     //寄存器堆读写
                   );
endmodule
```

Pipeline—Ex　执行模块接口

- **Pipeline_Ex**
  - 流水线CPU第三阶段
  - 执行指令所需求的操作
- **基本功能**
  - 执行是指对获取的操作数进行指令所指定的算数或逻辑运算
- **接口要求**
  - 执行模块接口如图：



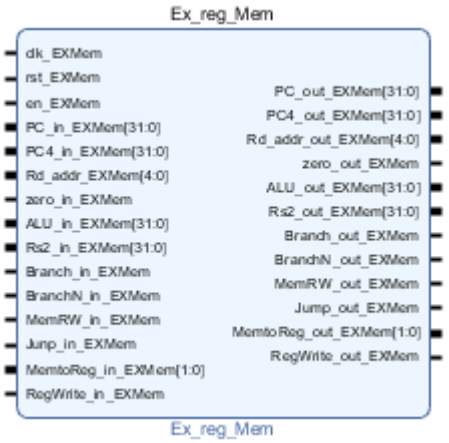
执行模块接口：Pipeline_Ex.v

```
module    Pipeline_Ex(
          input[31:0]   PC_in_EX，              //PC输入
          input[31:0]   Rs1_in_EX，             //操作数1输入
          input[31:0]   Rs2_in_EX,              //操作数2输入
          input[31:0]   Imm_in_EX ,             //立即数输入
          input         ALUSrc_B_in_EX ,        //ALU B选择
          input[2:0]    ALU_control_in_EX,      //ALU选择控制

          output reg  [31:0]  PC_out_EX,        //PC输出
          output reg  [31:0]  PC4_out_EX,       //PC+4输出
          output reg          zero_out_EX,      //ALU判0输出
          output reg  [31:0]  ALU_out_EX,       //ALU计算输出
          output reg  [31:0]  Rs2_out_EX        //操作数2输出
          )
endmodule
```

Pipeline—Ex_reg_Mem 执行-访存寄存器接口

- **Ex_reg_Mem**
  - 流水线CPU执行和访存之间的寄存器
  - 存储ALU数据和控制信号
- **基本功能**
  - 寄存EX级的输出指令，分割EX级和MEM级的指令或控制信号，防止相互干扰，在EX级执行结束时将指令的控制信号传递至下一级。
- **接口要求**
  - 执行访存寄存器接口如图：



执行-访存寄存器接口：Ex_reg_Mem.v

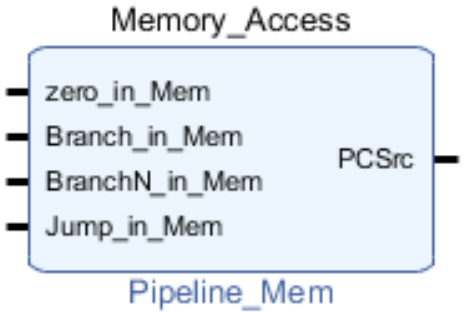
```
module    Ex_reg_Mem(
          input          clk_EXMem,          //寄存器时钟
          input          rst_EXMem,          //寄存器复位
          input          en_EXMem,           //寄存器使能
          input[31:0]    PC_in_EXMem,        //PC输入
          input[31:0]    PC4_in_EXMem,       //PC+4输入
          input  [4:0]   Rd_addr_EXMem,      //写目的寄存器地址输入
          input          zero_in_EXMem,      //zero
          input[31:0]    ALU_in_EXMem,       //ALU输入
          input[31:0]    Rs2_in_EXMem        //操作数2输入
          input          Branch_in_EXMem,    //Beq
          input          BranchN_in_EXMem,   //Bne
          input          MemRW_in_EXMem,     //存储器读写
          input          Jump_in_EXMem,      //Jal
          input [1:0]    MemtoReg_in_EXMem,  //写回
          input          RegWrite_in_EXMem,  //寄存器堆读写
```

执行-访存寄存器接口：Ex_reg_Mem.v

```
.....................................
        output reg[31:0]  PC_out_EXMem,          //PC输出
        output reg[31:0]  PC4_out_EXMem,         //PC+4输出
        output reg[4:0]   Rd_addr_out_EXMem,     //写目的寄存器输出
        output reg        zero_out_EXMem,        //zero
        output reg[31:0]  ALU_out_EXMem,         //ALU输出
        output reg[31:0]  Rs2_out_EXMem          //操作数2输出
        output reg        Branch_out_EXMem,      //Beq
        output reg        BranchN_out_EXMem,     //Bne
        output reg        MemRW_out_EXMem,       //存储器读写
        output reg        Jump_out_EXMem,        //Jal
        output reg        MemtoReg_out_EXMem,    //写回
        output reg        RegWrite_out_EXMem,    //寄存器堆读写
            );
endmodule
```

Pipeline—Mem　存储器访问模块接口

- **Pipeline_Mem**
  - 流水线CPU第四阶段
  - 访问存储器的操作
- **基本功能**
  - 存储器访问是指存储器访问指令将数据从存储器读出，或者写入存储器的过程
- **接口要求**
  - 存储器访问模块接口如图：



访存模块接口：Pipeline_Mem.v

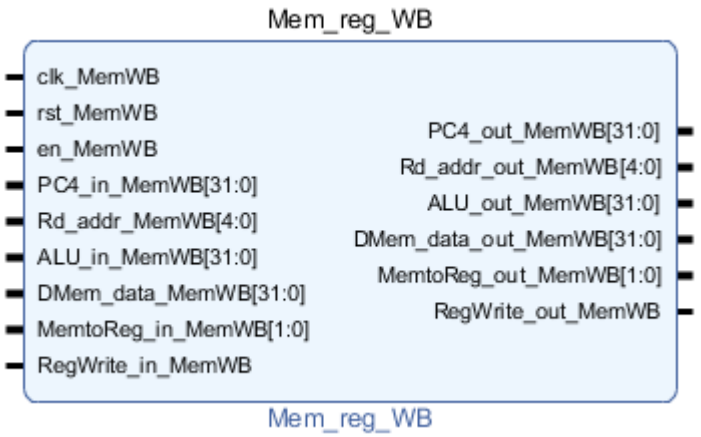
```
module      Pipeline_Mem(
        input       zero_in_Mem,        //zero
        input       Branch_in_Mem,      //beq
        input       BranchN_in_Mem,     //bne
        input       Jump_in_Mem,        //jal

        output      PCSrc,              //PC选择控制输出

)
endmodule
```

Pipeline—Mem_reg_WB　　访存-写回寄存器接口

- **Mem_reg_WB**
  - 流水线CPU访存和写回之间的寄存器
  - 存储ALU数据和存储器数据
- **基本功能**
  - 寄存Mem级的输出指令，以及输出数据，传递给写回阶段和寄存器堆。
- **接口要求**
  - 访存写回寄存器接口如图：

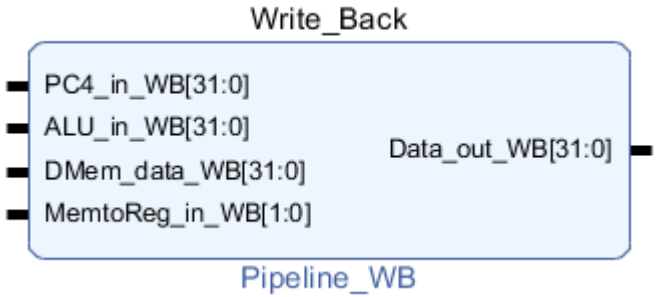


访存-写回寄存器接口:Mem_reg_WB.v

```
module    Mem_reg_WB(
          input              clk_MemWB,              //寄存器时
          input              rst_MemWB,              //寄存器复位
          input              en_MemWB,               //寄存器使能
          input[31:0]        PC4_in_MemWB,           //PC+4输入
          input[4:0]         Rd_addr_MemWB,          //写目的地址输入
          input[31:0]        ALU_in_MemWB,           //ALU输入
          input[31:0]        Dmem_data_MemWB         //存储器数据输入
          input[1:0]         MemtoReg_in_MemWB,      //写回
          input              RegWrite_in_MemWB,      //寄存器堆读写
          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);
                    ……
endmodule
```

Pipeline—WB　写回模块接口

- **Pipeline_WB**
  - 流水线CPU第五阶段
  - 结果写回寄存器堆的操作
- **基本功能**
  - 写回是指将指令执行的结果写回寄存器堆的过程；如果是普通运算指令，该结果值来源于‘执行’阶段计算的结果；如果是LOAD指令，该结果来源于‘访存’阶段从存储器读取出来的数据；如果是跳转指令，该结果来源于PC+4。
- **接口要求**
  - 写回模块接口如图：

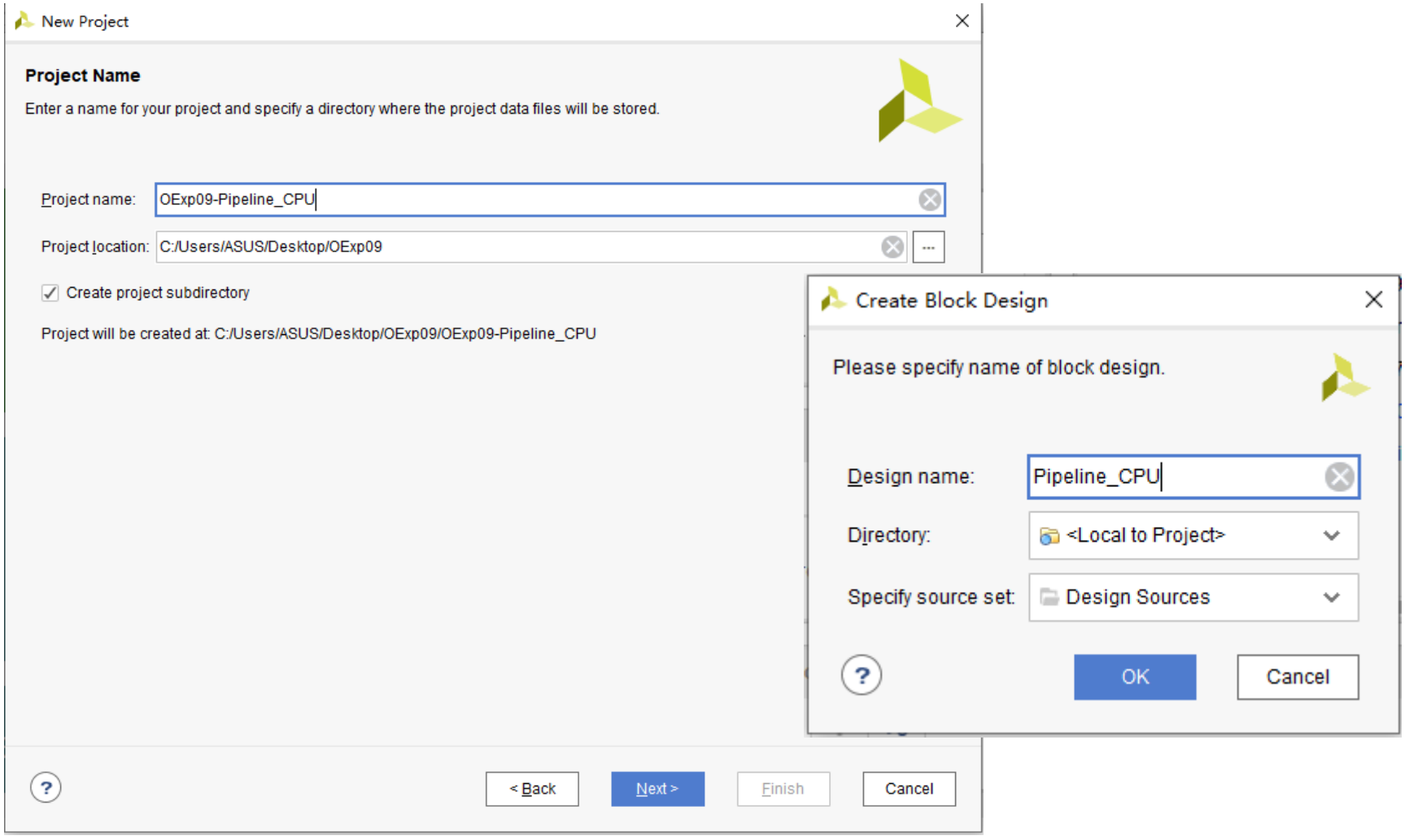


访存模块接口：Pipeline_WB.v

```
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
);
  ……
endmodule
```

流水线CPU集成


New Project
Project Name
Enter a name for your project and specify a directory where the project data files will be stored.
Project name: OExp09-Pipeline_CPU
Project location: C:/Users/ASUS/Desktop/OExp09
Create project subdirectory
Project will be created at: C:/Users/ASUS/Desktop/OExp09/OExp09-Pipeline_CPU
< Back
Next >
Finish
Cancel
Create Block Design
Please specify name of block design.
Design name: Pipeline_CPU
Directory: <Local to Project>
Specify source set: Design Sources
OK
Cancel

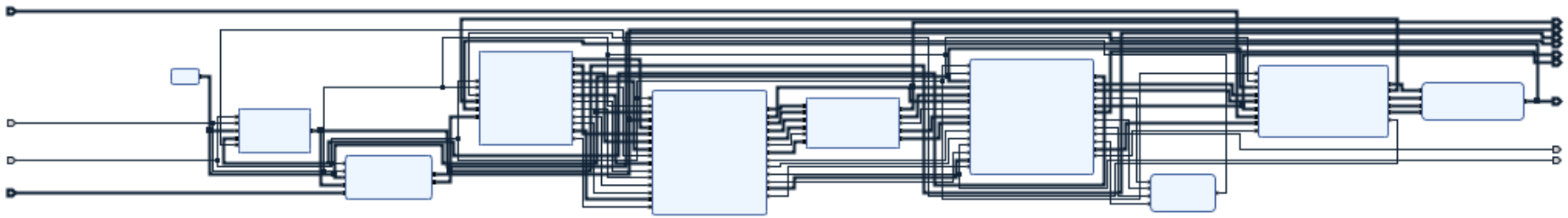
设计要点

拷贝下列模块到Pipeline_CPU工程目录：

Pipeline_IF、IF_reg_ID、Pipeline_ID、ID_reg_Ex、Pipeline_Ex、Ex_reg_Mem、Pipeline_Mem、Mem_reg_WB、Pipeline_WB

添加模块路径到Pipeline_CPU工程目录：

流水线CPU集成

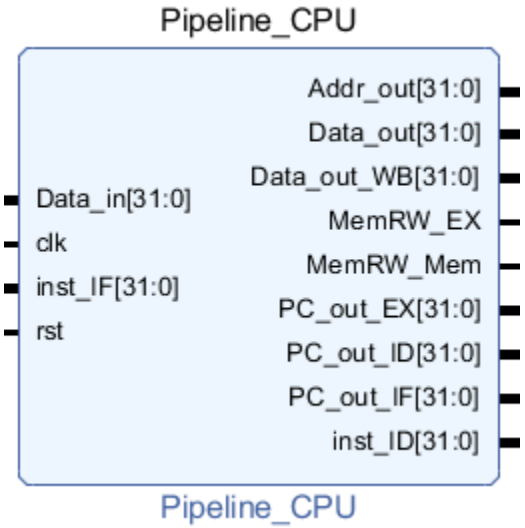


集成Pipeline CPU的模块层次结构

五级模块、四个寄存器

流水线CPU集成

- 替换 Exp04的单周期CPU为本实验集成的五级流水线CPU



Pipeline CPU接口：Pipeline_CPU.v

```
module   Pipeline_CPU(
            input              clk,          //时钟
            input              rst,          //复位
            input[31:0]        Data_in,      //存储器数据输入
            input[31:0]        inst_IF,      //取指阶段指令
            output [31:0]      PC_out_IF,    //取指阶段PC输出
            output [31:0]      PC_out_ID,    //译码阶段PC输出
            output [31:0]      inst_ID,      //译码阶段指令
            output [31:0]      PC_out_Ex,    //执行阶段PC输出
            output [31:0]      MemRW_Ex,     //执行阶段存储器读写
            output [31:0]      MemRW_Mem,    //访存阶段存储器读写
            output [31:0]      Addr_out,     //地址输出
            output [31:0]      Data_out,     //CPU数据输出
            output [31:0]      Data_out_WB   //写回数据输出
);  ……
endmodule
```

若想观察其他信号，则在封装CPU时将对应端口输出即可

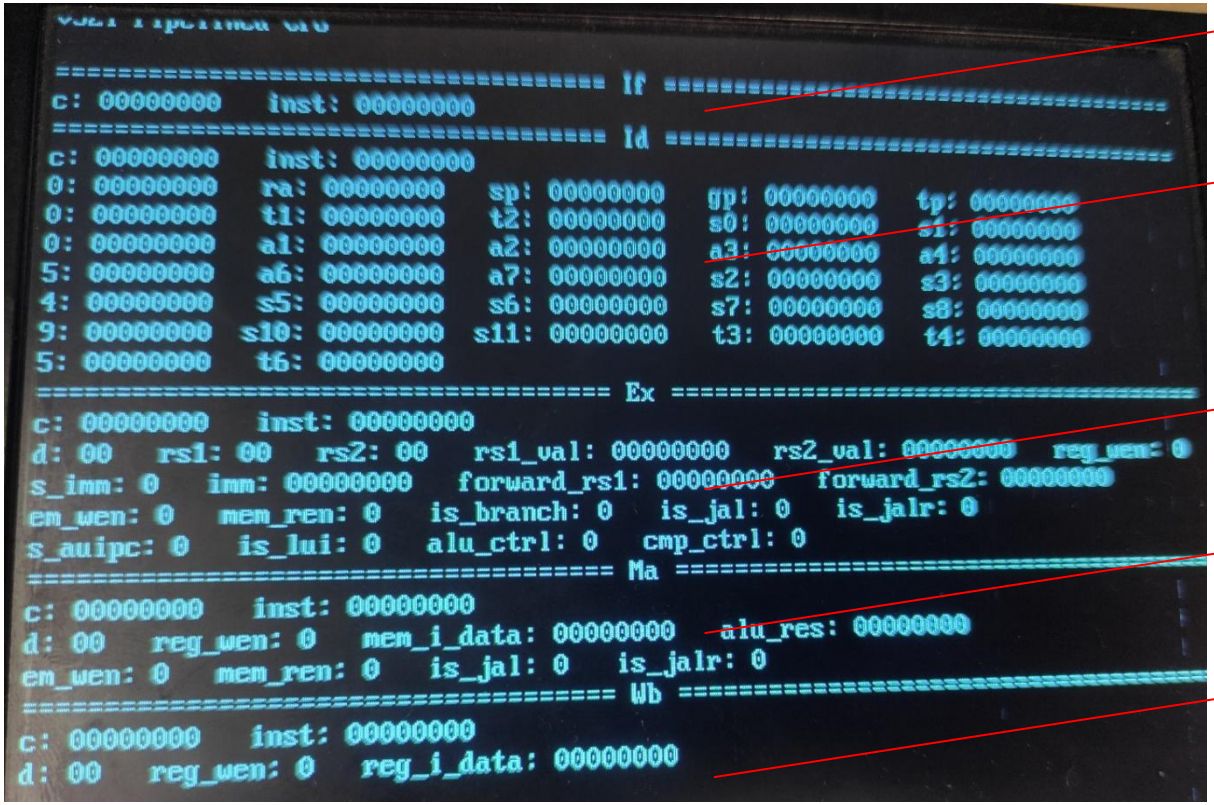
- 任务二：设计流水线测试方案并完成测试

物理验证

- **使用DEMO程序目测CPU运行情况**
  - DEMO接口功能
    - SW[8]=0， SW[2]=0(全速运行)
    - SW[8]=0， SW[2]=1(自动单步)
    - SW[8]=1， SW[2]=x(手动单步)
- **用汇编语言设计测试程序**
  - 测试ALU指令(R-格式译码\I-立即数格式译码)
  - 测试LW指令(I-格式译码)
  - 测试SW指令(S-格式译码)
  - 测试分支指令(B-格式译码)

物理验证

- 为更好追踪流水线CPU的特点，VGA显示的接口稍有调整，分别从取指、译码、执行、访存、写回进行显示，请采用更新版本的IP
- 实验中选取了部分信号进行观测，若想观察其他信号，请参照Lab04将其他待测信号引出即可



取指阶段信号

译码阶段信号

执行阶段信号

访存阶段信号

写回阶段信号

测试程序参考：

- 方案一： 无冒险的流水线测试（p.mem）

```
#baseAddr 0000
main:   addi x1,x0,0x1        #x1 = 0x1
        addi x2,x0,0x1        #x2 = 0x1
        addi x3,x0,0x1        #x3 = 0x1
        addi x4,x0,0x1        #x4 = 0x1
        lw x5,0x8(x0)      #x5 = 0x80000000
        add x6,x1,x1          #x6 = 0x2
        xor x7,x1,x2          #x7 = 0
        sub x8,x2,x1          #x8 = 0
        ori x9,x3,-1        #x9 = 0xFFFFFFFF
        and x10,x4,x3         #x10= 0x2
        sw x5,0x4(x0)         #mem(1)=
                              0x80000000
        slt x11,x6,x5       #x11= 0x1
        xori x12,x7,0xAA    #x12= 0xAA
        srl x13,x5,x1      #X13=0x40000000
        andi x14,x8,0x1     #x14= 0x1
        or x15,x9,x3       #x15=0xFFFFFFFF
        add x16,x10,x10     #x16= 0x4
        xor x17,x11,x8      #x17= 0x1
        lw x18,0x4(x0)   #x18=0x80000000
        slt x19,x12,x4      #x19= 0
        srli x20,x13,0x1    #x20= 0x20000000
        and x21,x14,x6      #x21= 0
        sub x22,x5,x1       #x22= 0x7FFFFFFF
        addi x23,x10,0x1    #x23= 0x3
        or x24,x16,x9       #x24= 0xFFFFFFFB
        xor x25,x19,x11     #x25= 0x1
        andi x26,x20,0xFF   #x26= 0x200000FF
        add x27,x18,x3      #x27= 0x80000001
        srl x28,x20,x2      #x28= 0x10000000
        ori x29,x19,0xAF    #x29= 0xAF
        add x30,x20,x1      #x30= 0x20000001
        lw x31,0x8(x0)      #x31= 0x80000000
        jal x0,main
        add x0,x0,x0
        add x0,x0,x0
        add x0,x0,x0
```

执行顺序如何？

测试程序参考：

- 方案二： 有冒险的流水线测试(h.mem)

```
#baseAddr 0000
main:   addi x1,x0,0x1      #x1 = 0x1
        addi x2,x0,0x1      #x2 = 0x1
        addi x3,x0,0x1      #x3 = 0x1
        addi x4,x0,0x1      #x4 = 0x1
        lw x5,0x8(x0)       #x5 = 0x80000000
        add x6,x5,x1        #x6 = 0x80000001
        xor x7,x1,x2        #x7 = 0
        sub x8,x1,x7        #x8 = 0x1
        ori x9,x3,-1        #x9 = 0xFFFFFFFF
        and x10,x4,x3       #x10= 0x1
        sw x5,0x4(x0)   #mem(1)=0x8000000
        slt x11,x6,x5       #x11= 0x0
        xori x12,x7,0xAA    #x12= 0xAA
        beq x3,x8,loop1
        addi x0,x0,0x0
        add x0,x0,x0
loop1:  srl x13,x5,x1       #x13= 0x40000000
        andi x14,x8,0x1     #x14= 0x1
        or x15,x9,x3        #x15= 0xFFFFFFFF
        add x16,x10,x10     #x16= 0x2
        xor x17,x11,x8      #x17= 0x1
        lw x18,0x4(x0)      #x18= 0x80000000
        slt x19,x12,x4      #x19= 0
        srli x20,x13,0x1    #x20= 0x20000000
        and x21,x14,x10     #x21= 0x1
        bne x14,x12,loop2
        addi x0,x0,0x0
loop2:sub x22,x5,x1 #x22= 0x7FFFFFFF
        addi x23,x10,0x1    #x23= 0x2
        or x24,x16,x9       #x24= 0xFFFFFFFF
        xor x25,x19,x11     #x25= 0x0
        andi x26,x20,0xFF   #x26= 0x200000FF
        add x27,x18,x3      #x27= 0x80000001
        srl x28,x20,x2      #x28= 0x10000000
        ori x29,x19,0xAF    #x29= 0xAF
        add x30,x20,x1      #x30= 0x20000001
        lw x31,0x8(x0)      #x31= 0x80000000
        jal x0,main
        add x0,x0,x0
        add x0,x0,x0
        add x0,x0,x0
```

结果是否正确？

设计测试记录表格

- **ALU指令**测试结果记录
  - 自行设计记录表格


END