

Lab05-3

流水线处理器—Ex、Mem、WB 设计与集成

Ma De (马德)

made@zju.edu.cn

2025

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. 理解流水线执行、存储器访问、写回的原理
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

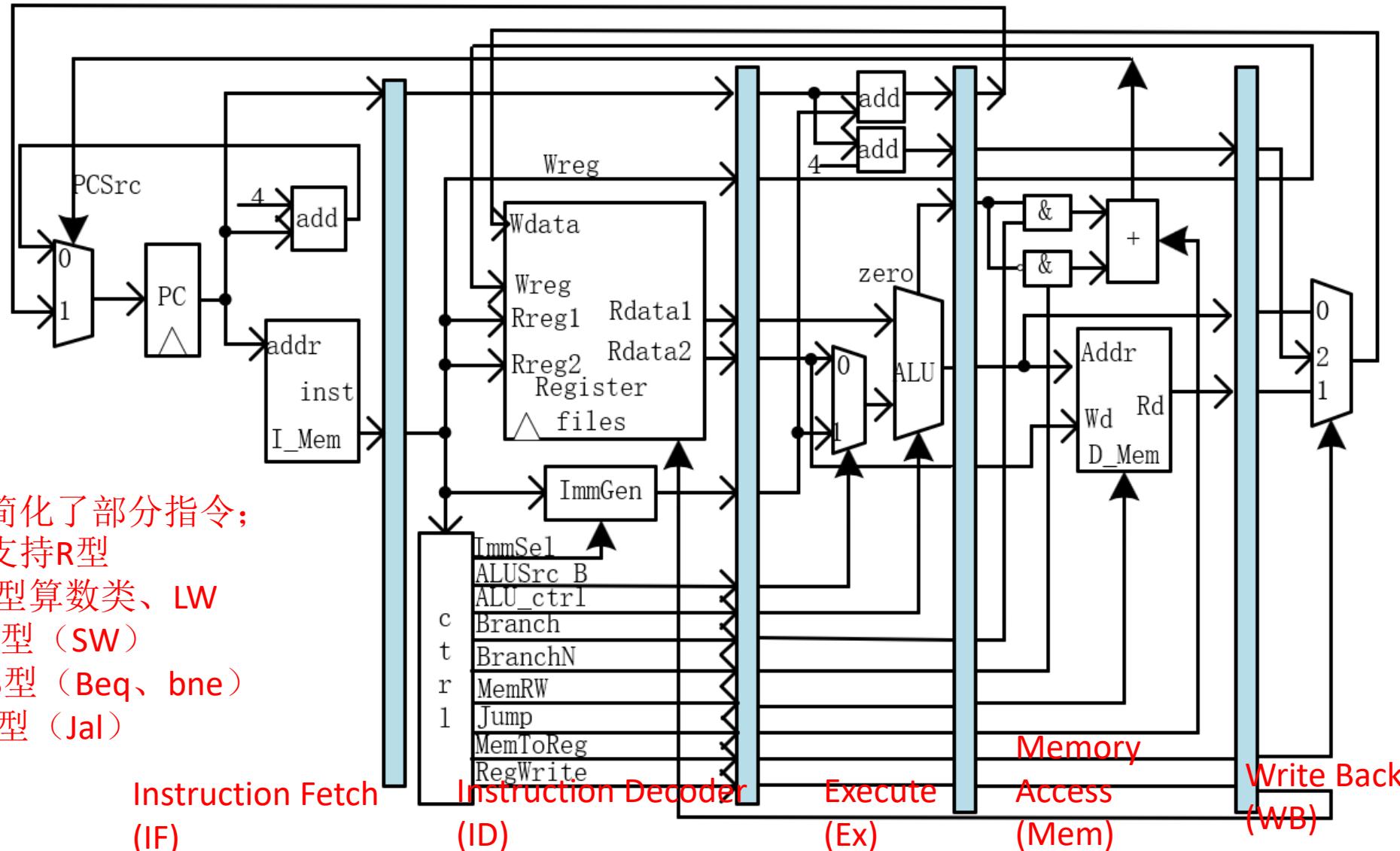
实验目标及任务

- **目标：**熟悉RISC-V五级流水线的工作特点，了解执行、存储器访问、写回的原理，掌握IP核的使用方法，集成并测试CPU
- **任务一：**设计执行（Ex）、存储器访问（Mem）、写回（WB）模块，替换lab04的流水线CPU并完成集成
 - 设计执行模块，替换OExp05-2的执行模块并完成集成
 - 设计访存模块，替换OExp05-2的访存模块并完成集成
 - 设计写回模块，替换OExp05-2的写回模块并完成集成
- **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

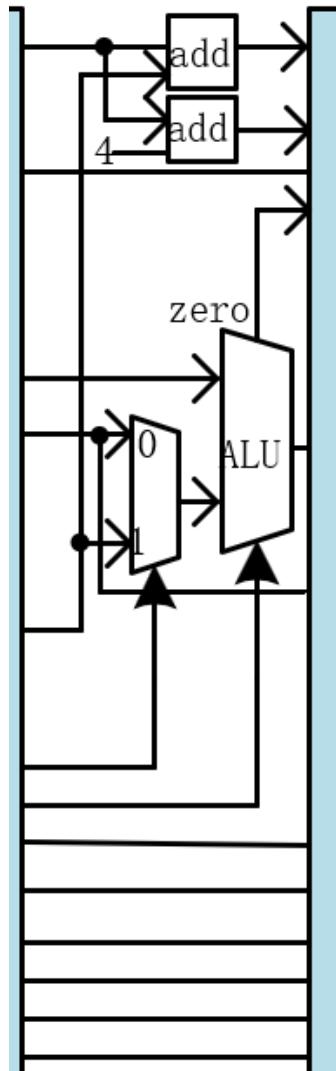
- 执行（Ex）模块介绍
- 访存（Mem）模块介绍
- 写回（WB）模块介绍

Pipelined RISC-V RV32I Datapath



执行—功能介绍

Execute(Ex)

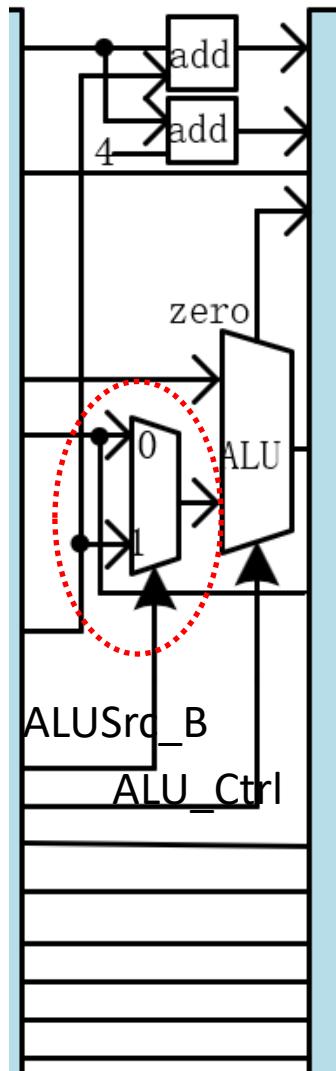


- 执行：执行阶段涉及运算单元（ALU）它获取操作数并完成指定的算数运算或逻辑运算。
- **Ex_reg_Mem**: 暂存运算结果和控制信号，以待下一级使用

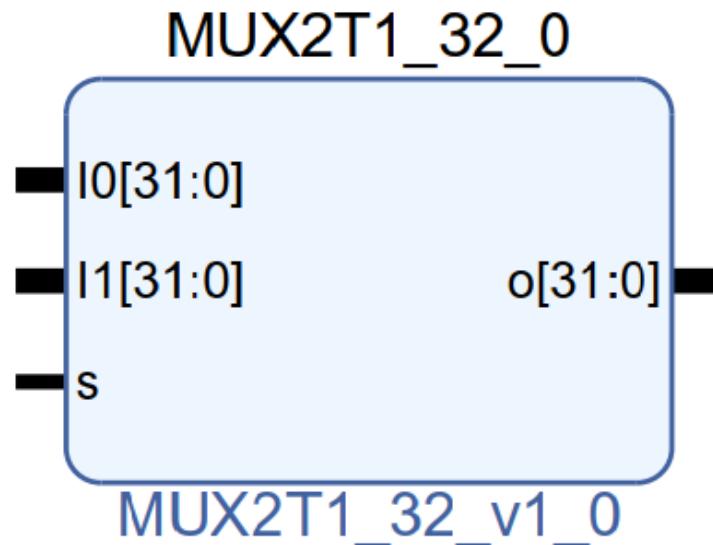
Ex_reg_Mem
寄存器

执行—部件介绍

■ 多选器 (MUX2T1_32)



- **多选器 (MUX2T1_32) :** 本质上是一个32位二选一的多路器，用于选择 ALU 的第二个操作数，当 ALUSrc_B=0; 选择寄存器堆数据2, 当ALUSrc_B=1; 选择立即数。



执行—部件介绍

■ 多选器 (MUX2T1_32)

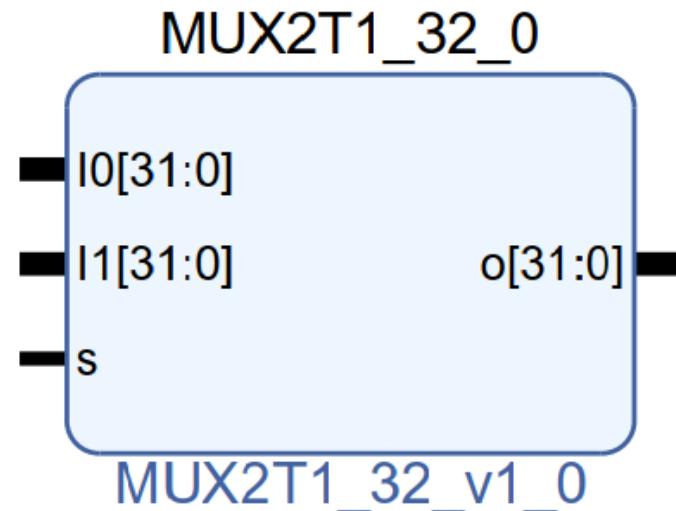
■ 模块名: MUX2T1_32

- 数据输入: I0(31:0)
- 数据输入: I1(31:0)
- 数据输入: S
- 数据输出: O(31:0)

■ 参考描述结构

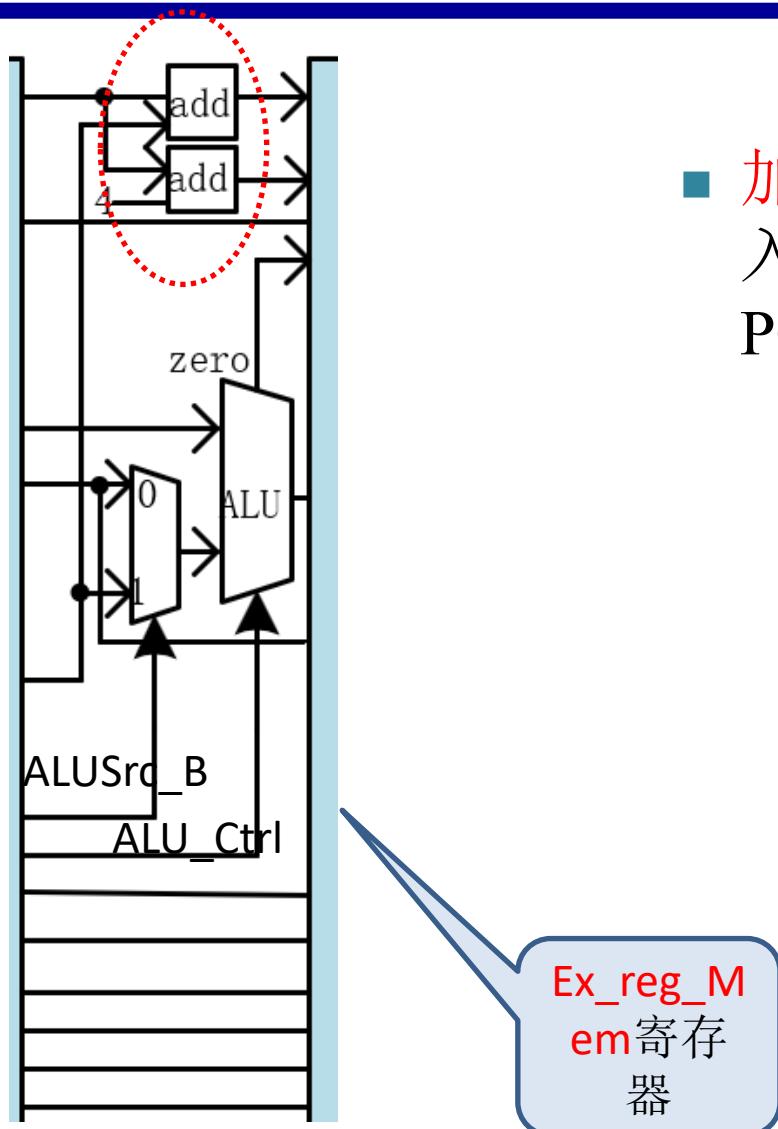
```
module MUX2T1_32(input [31:0] I0,  
                   input [31:0] I1,  
                   input sel,  
                   output reg[31:0] o  
);  
    .....  
endmodule
```

可直接调用
OExp01的多
选器模块

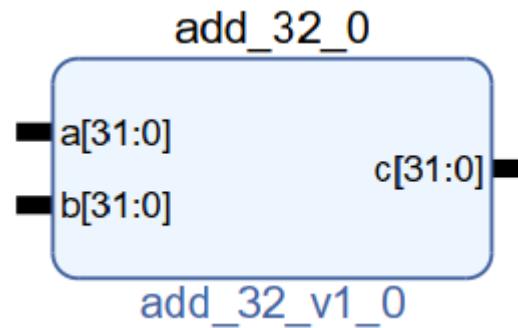


执行—部件介绍

■ 加法器 (add)



- 加法器 (add)：本质是一个二输入32位的加法器，一个用于计算PC+4、一个用于计算PC+Imm。



执行—部件介绍

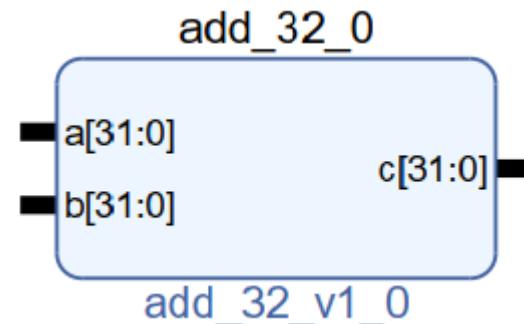
■ 加法器 (add)

- 模块名: add_32
 - 数据输入: a(31:0)
 - 数据输入: b(31:0)
 - 数据输出: c(31:0)

- 参考描述结构

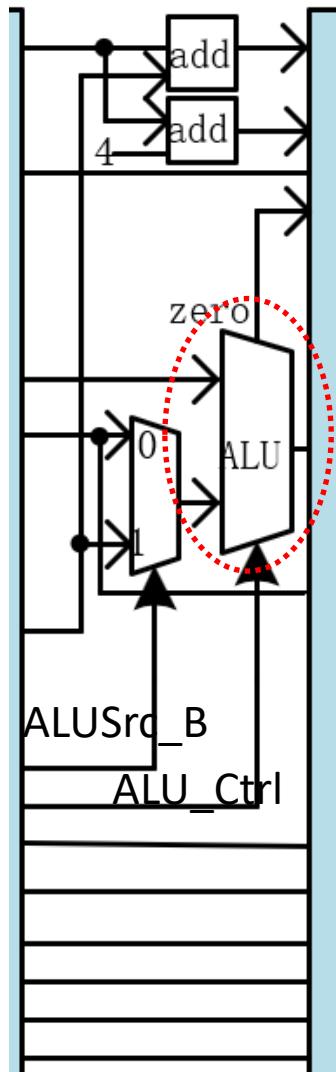
```
module add_32(input [31:0] a,  
               input [31:0] b,  
               output [31:0] c  
);  
    .....  
endmodule
```

可直接调用
OExp01的加
法器模块



执行—部件介绍

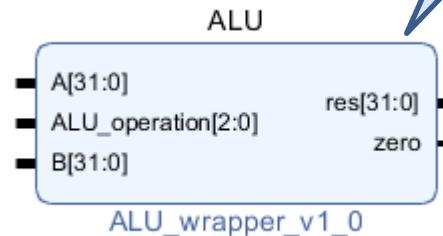
■ ALU



- **ALU**: 本质是一个二输入32位的运算器，实现算数运算和逻辑运算

-

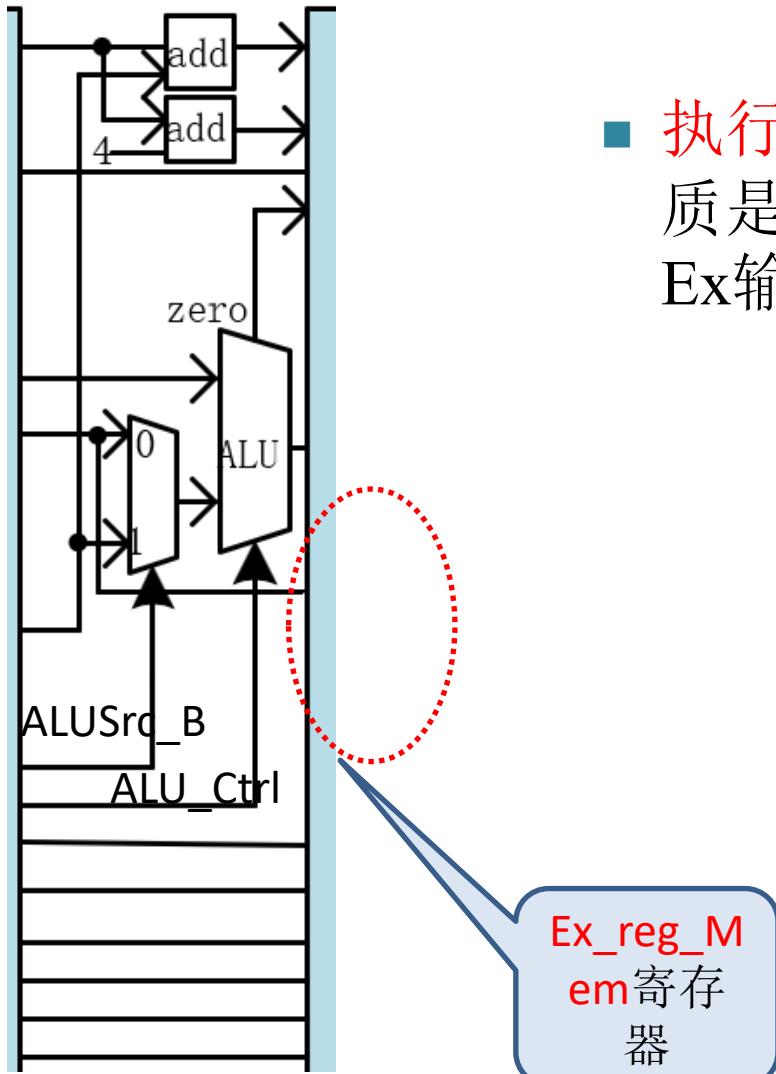
可直接调用
OExp04的
ALU模块



Ex_reg_M
em寄存器

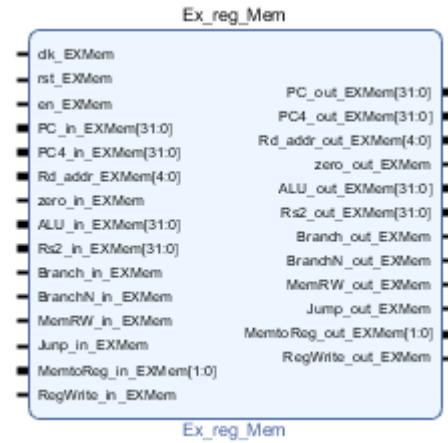
执行—部件介绍

■ 执行-访存寄存器 (Ex_Reg_Mem)



- 执行-访存寄存器 (Ex_Reg_Mem)：本质是一个寄存器，用于寄存控制信号和 Ex 输出结果。

本实验设计



执行—部件介绍

■ 执行-访存寄存器 (Ex_reg_Mem)

◎ Ex_reg_Mem

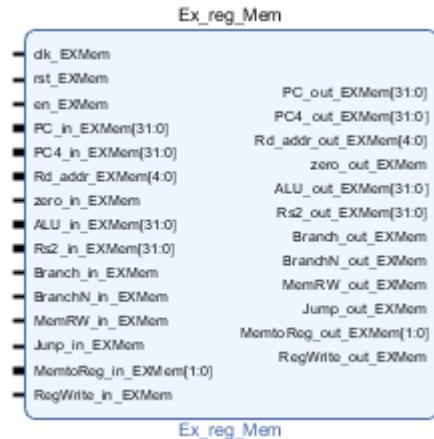
- 流水线CPU执行和访存之间的寄存器
- 存储ALU数据和控制信号

◎ 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。

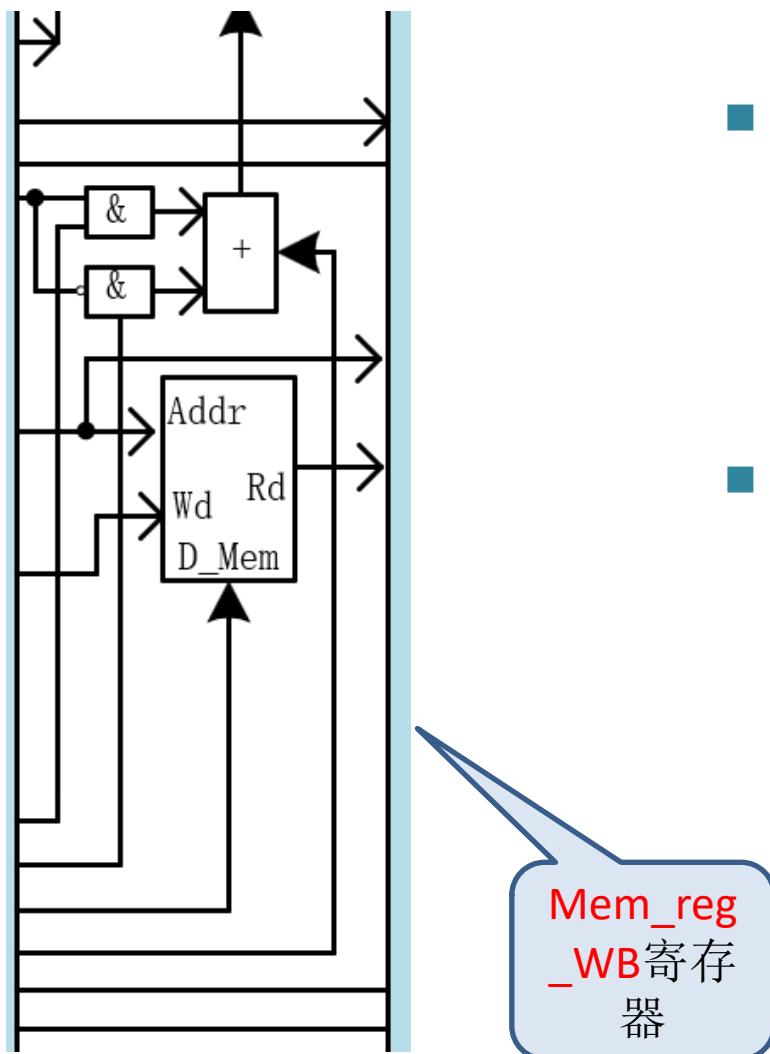
◎ 接口要求

- 执行访存寄存器接口如图：



访存----功能介绍

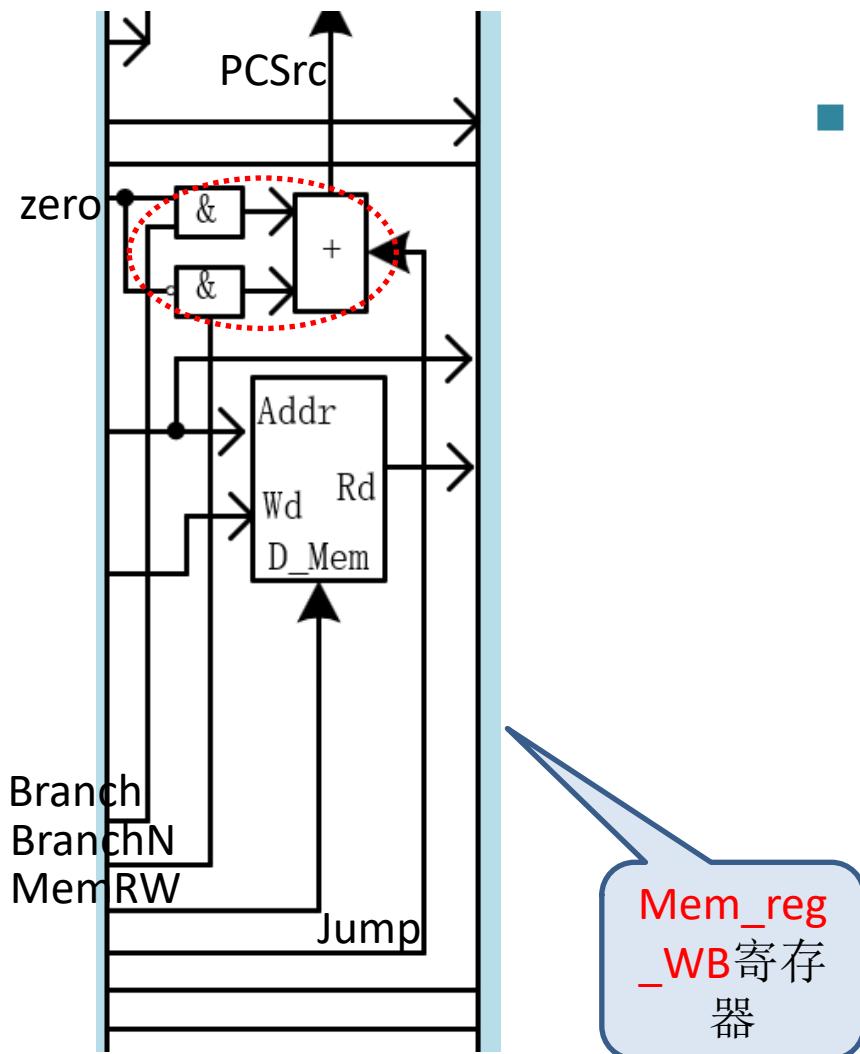
■ Memory Access(Mem)



- **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

访存----部件介绍

■ 逻辑单元

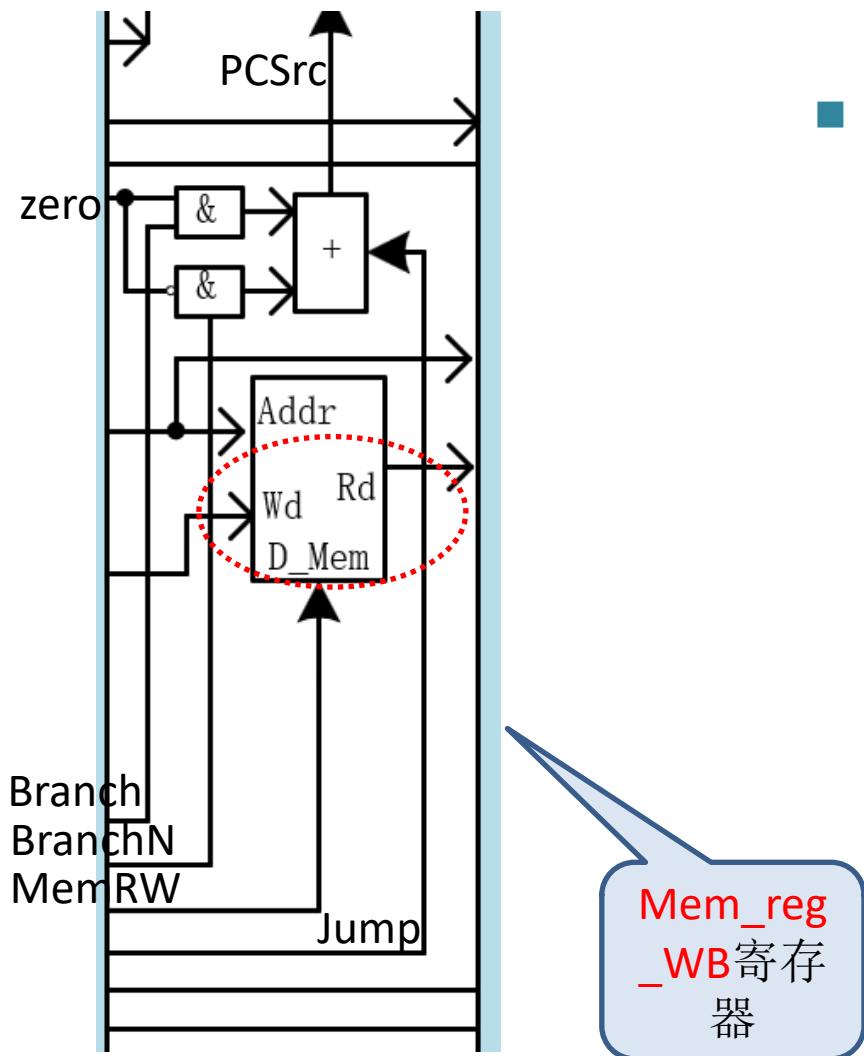


- 逻辑单元：两输入与门和三输入或门组成，用于判断分支和跳转是否发生，生成PCSrc选择控制PC值。

$$(\text{PCSrc} = (\text{Branch} \& \text{zero}) | (\text{BranchN} \& (\sim \text{zero})) | \text{Jump})$$

访存----部件介绍

■ 数据存储器 (D_Mem)



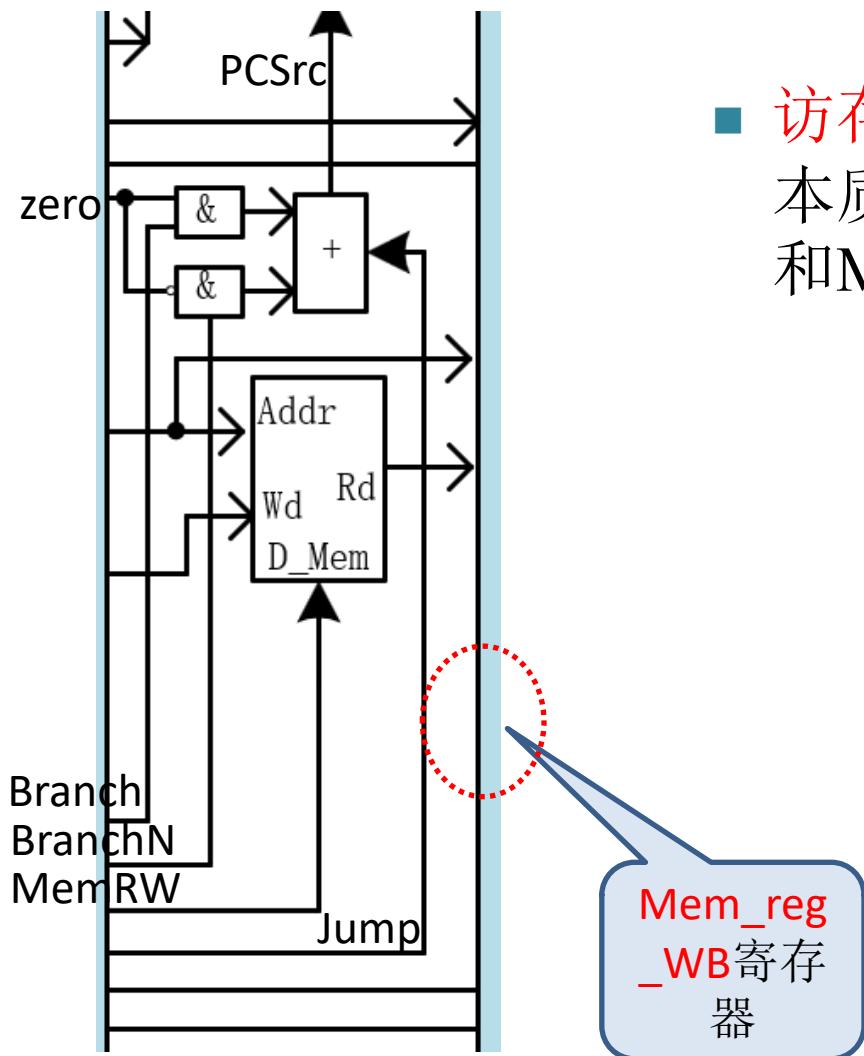
- **数据存储器 (D_Mem) :** 本质是一个RAM可读可写, LOAD、STORE指令的操作对象 (D_Mem 不是CPU的内部部件, 只在构建 SOC系统时才用到)。

可直接仿照
OExp01由IP
工具生成

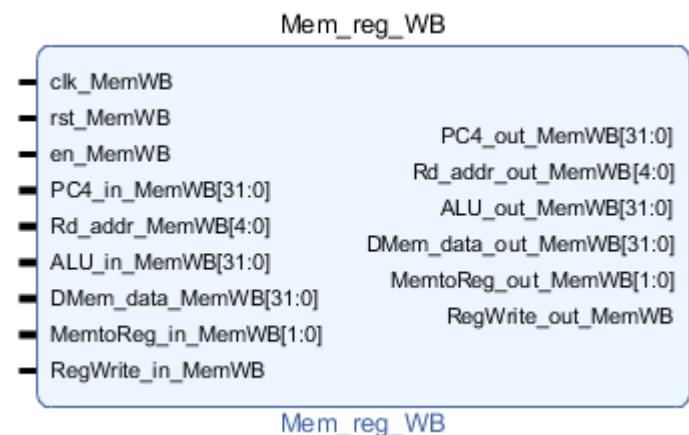
Mem_reg
_WB寄存
器

访存----部件介绍

■ 访存-写回寄存器 (Mem_reg_WB)



- 访存-写回寄存器 (Mem_Reg_WB)：
本质是一个寄存器，用于寄存控制信号和Mem结果输出。



访存—部件介绍

■ 访存-写回寄存器 (Mem_reg_WB)

◎ Mem_reg_WB

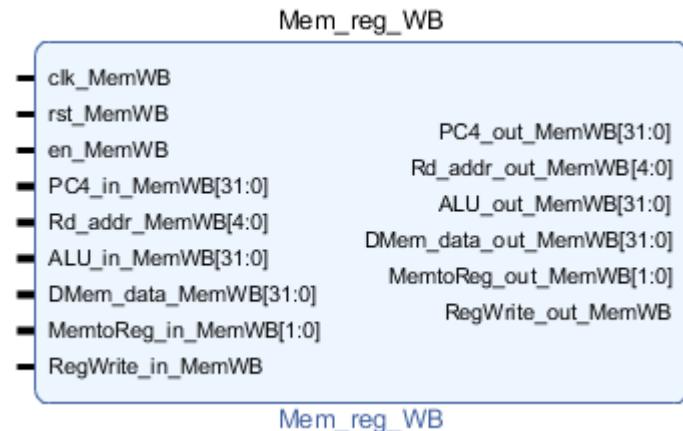
- 流水线CPU访存和写回之间的寄存器
- 存储ALU数据和存储器数据

◎ 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。

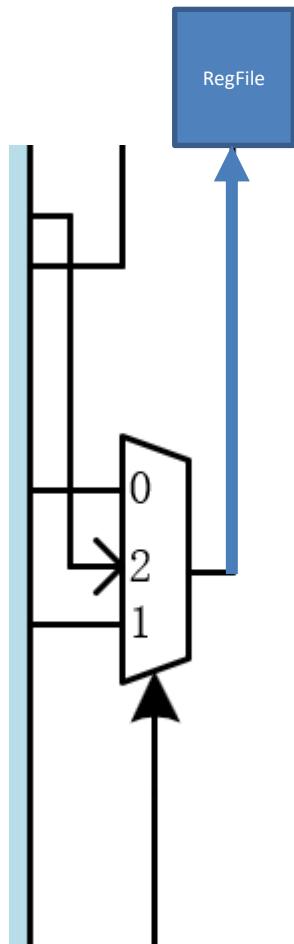
◎ 接口要求

- 访存写回寄存器接口如图：



写回—功能介绍

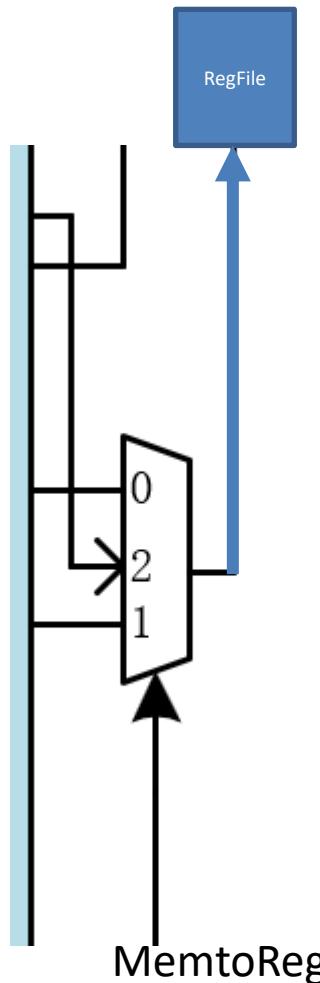
■ Write Back(WB)



- **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。

写回—部件介绍

■ 多选器 (MUX4T1_32)



- **多选器 (MUX4T1_32)**：本质是一个32位四选一的多路器，用于选择写回寄存器堆的数据，
- 当 $\text{MemtoReg}=0$ ；选择ALU输出，
- 当 $\text{MemtoReg}=1$ ；选择存储器输，
- 当 $\text{MemtoReg}=2$ ；选择 $\text{PC}+4$ 。

-
- **任务一：**设计执行（Ex）、存储器访问（Mem）、写回（WB）模块，替换Lab05-2的流水线CPU并完成集成
 - 设计执行模块，替换OExp05-2的执行模块并完成集成
 - 设计存储器访问模块，替换OExp05-2的存储器访问模块并完成集成
 - 设计写回模块，替换OExp05-2的写回模块并完成集成

执行模块、Ex_reg_Mem寄存器 设计集成

Pipeline—Ex 执行模块接口

○ Pipeline_Ex

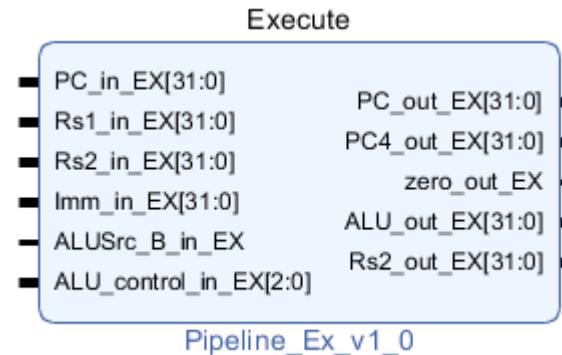
- ④ 流水线CPU第三阶段
- ④ 执行指令所需求的操作

○ 基本功能

- ④ 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

- ④ 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
```

执行模块设计

New Project

Project Name
Enter a name for your project and specify a directory where the project data files will be stored.

Project name: Pipeline_Ex

Project location: C:/Users/ASUS/Desktop/stall/IP/CPU/pipeline

Create project subdirectory
Project will be created at: C:/Users/ASUS/Desktop/stall/IP/CPU/pipeline/Pipeline_Ex

Create Block Design

Please specify name of block design.

Design name: Pipeline_Ex

Directory: <Local to Project>

Specify source set: Design Sources

OK Cancel

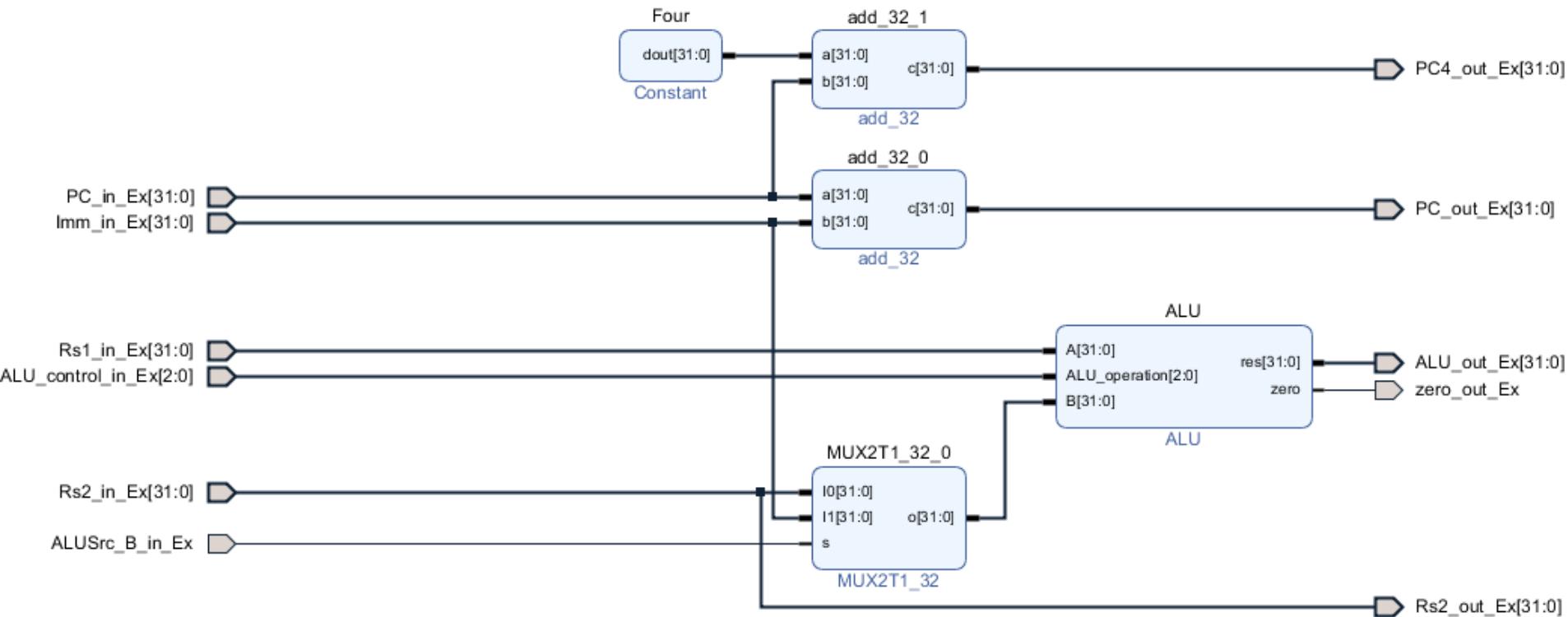
< Back Next > Finish

执行模块设计

拷贝下列模块到Execute工程目录：
MUX2T1_32、alu、add_32

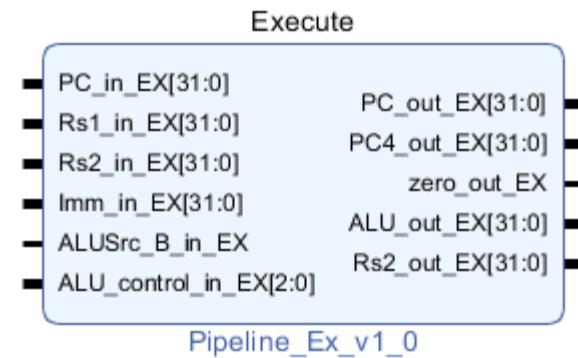
添加模块路径到Execute工程目录：

执行模块设计



执行模块设计

- ✓ Pipeline_Ex (Pipeline_Ex.bd) (5)
 - ✓ Pipeline_Ex_ALU_wrapper_0_0 (Pipeline_Ex_ALU_wrapper_0_0.xci)
 - ✓ Pipeline_Ex_MUX2T1_32_0_0 (Pipeline_Ex_MUX2T1_32_0_0.xci)
 - ✓ Pipeline_Ex_add_32_0_0 (Pipeline_Ex_add_32_0_0.xci)
 - ✓ Pipeline_Ex_add_32_0_1 (Pipeline_Ex_add_32_0_1.xci)
 - ✓ Pipeline_Ex_xlconstant_0_0 (Pipeline_Ex_xlconstant_0_0.xci)



○ Ex_reg_Mem

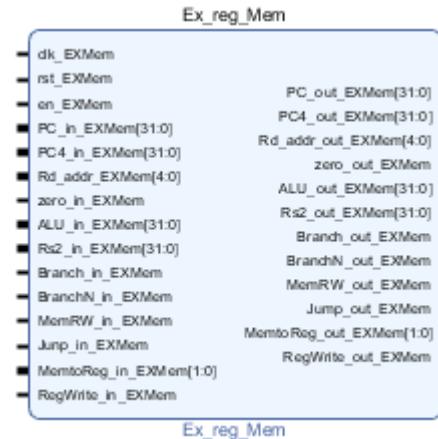
- 流水线CPU执行和访存之间的寄存器
- 存储ALU数据和控制信号

○ 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
```

访存模块、Mem_reg_WB寄存器 设计集成

○ Pipeline_Mem

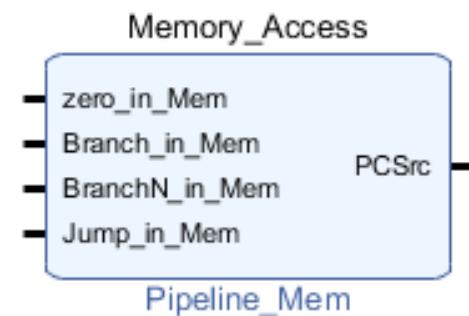
- ④ 流水线CPU第四阶段
- ④ 访问存储器的操作

○ 基本功能

- ④ 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

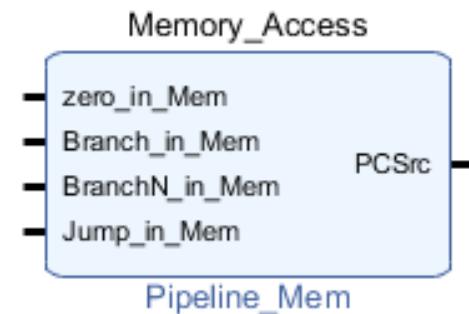
○ 接口要求

- ④ 存储器访问模块接口如图：

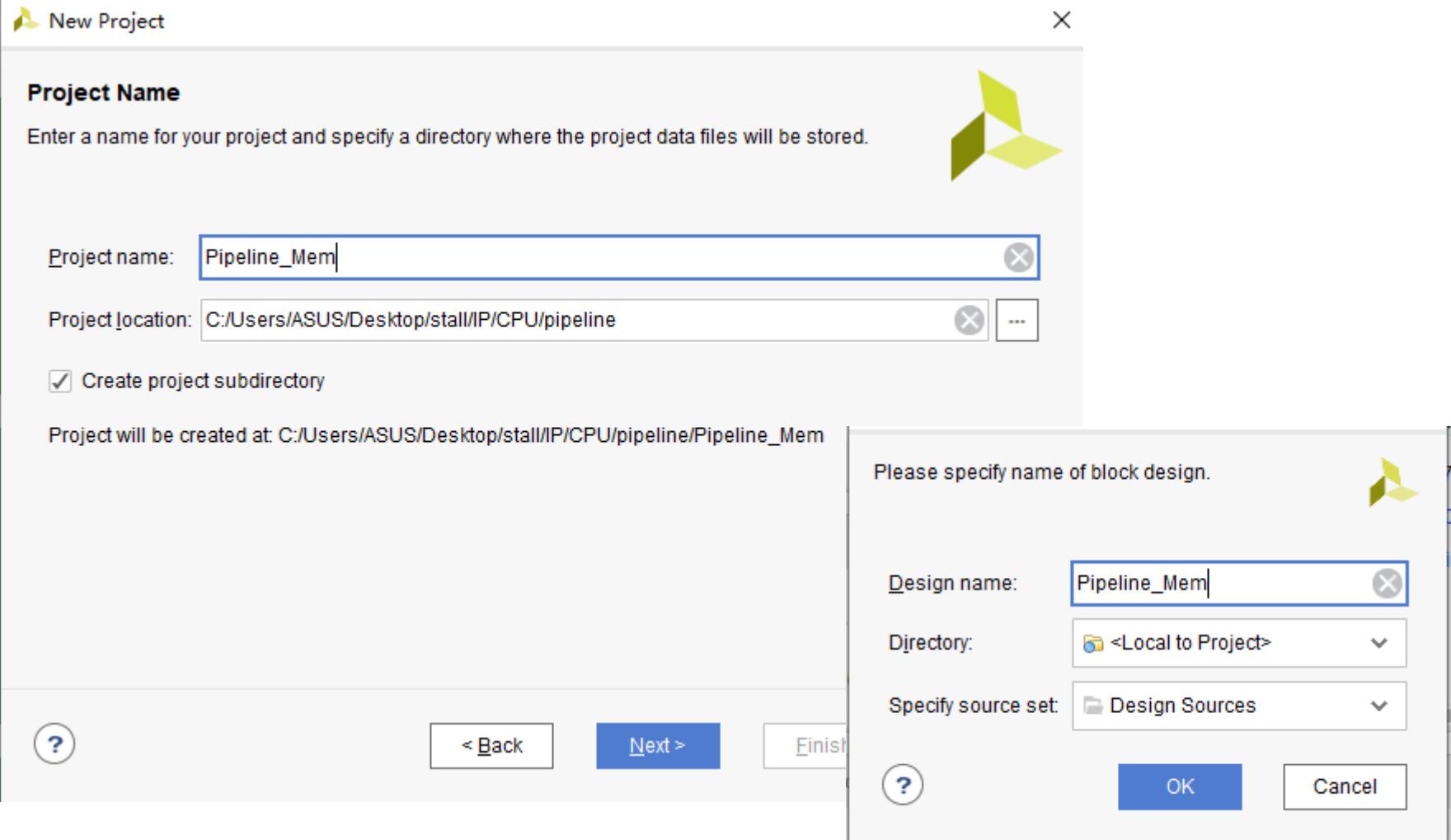


访存模块接口： Pipeline_Mem.v

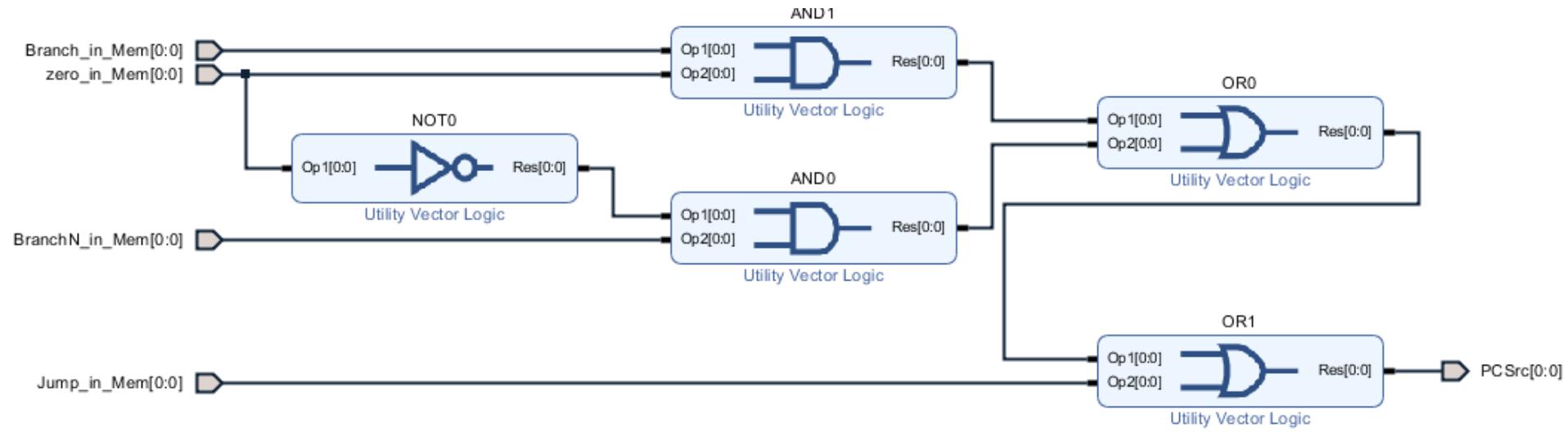
```
module Pipeline_Mem(  
    input zero_in_Mem,           //zero  
    input Branch_in_Mem,         //beq  
    input BranchN_in_Mem,        //bne  
    input Jump_in_Mem,          //jal  
  
    output PCSrc,               //PC选择控制输出  
)  
endmodule
```



访存模块设计



访存模块设计



◎ Mem_reg_WB

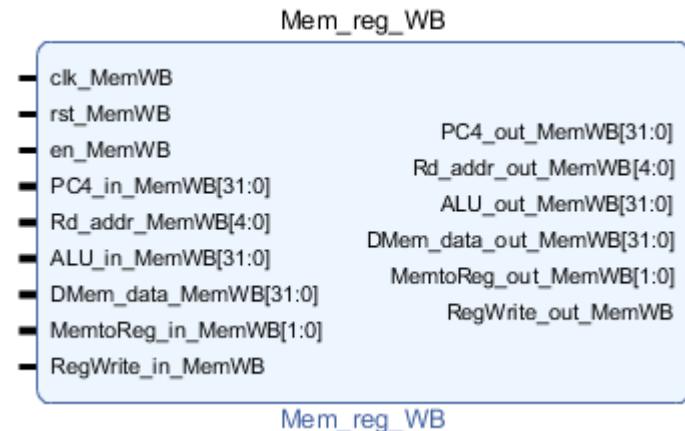
- 流水线CPU访存和写回之间的寄存器
- 存储ALU数据和存储器数据

◎ 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:Memm_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
```

写回模块设计集成

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

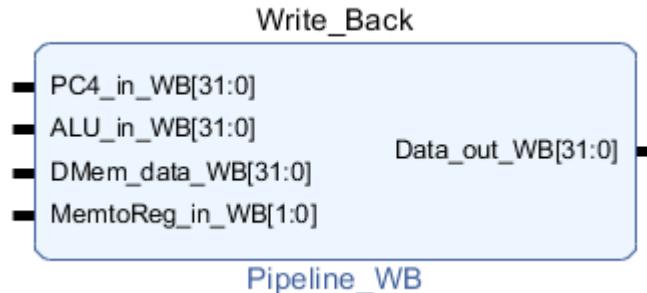
⌚ 写回是指将指令执行的结果写回寄存器堆的过程；如果是普通运算指令，该结果值来源于‘执行’阶段计算的结果；如果是LOAD指令，该结果来源于‘访存’阶段从存储器读取出来的数据；如果是跳转指令，该结果来源于PC+4。

◎ 接口要求

- ⌚ 写回模块接口如图：

写回模块接口： 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
```



写回模块设计

New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: Pipeline_WB

Project location: C:/Users/ASUS/Desktop/stall/IP/CPU/pipeline

Create project subdirectory

Project will be created at: C:/Users/ASUS/Desktop/stall/IP/CPU/pipeline/Pipeline_WB

< Back Next > Finish

Create Block Design

Please specify name of block design.

Design name: Pipeline_WB

Directory: <Local to Project>

Specify source set: Design Sources
指定源文件组

OK Cancel

This image shows two windows from a software interface. The main window is titled 'New Project' and contains fields for 'Project name' (Pipeline_WB) and 'Project location' (C:/Users/ASUS/Desktop/stall/IP/CPU/pipeline). It also has a checked checkbox for 'Create project subdirectory'. Below these fields, it displays the full path where the project will be created: C:/Users/ASUS/Desktop/stall/IP/CPU/pipeline/Pipeline_WB. At the bottom of this window are buttons for '?', '< Back' (disabled), 'Next >', and 'Finish'. The second window, titled 'Create Block Design', asks for a 'Design name' (Pipeline_WB) and a 'Directory' (<Local to Project>). It also includes a 'Specify source set' dropdown set to 'Design Sources' with an additional option '指定源文件组' highlighted. This window has buttons for '?', 'OK', and 'Cancel'.

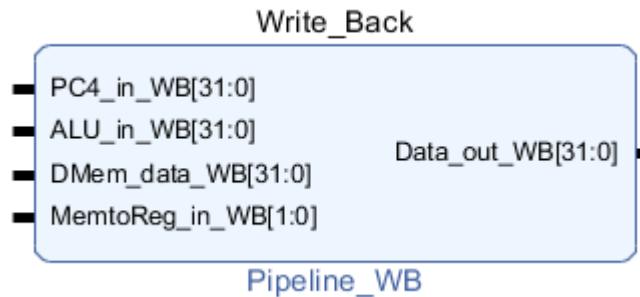
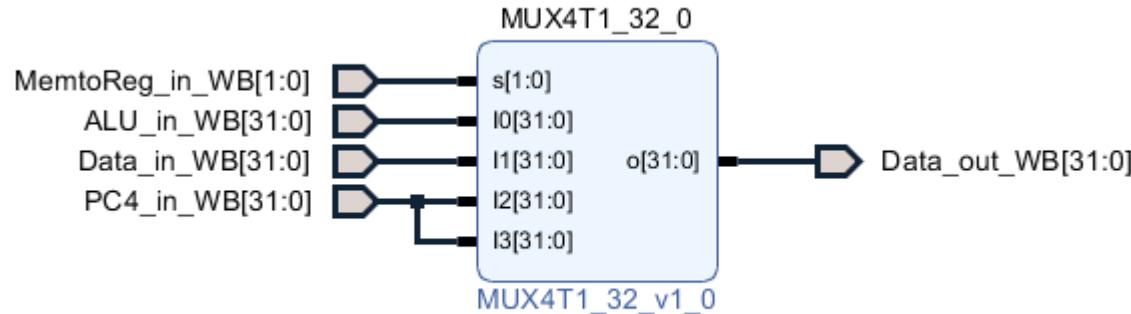
写回模块设计

拷贝下列模块到Write_Back工程目录：

MUX4T1_32

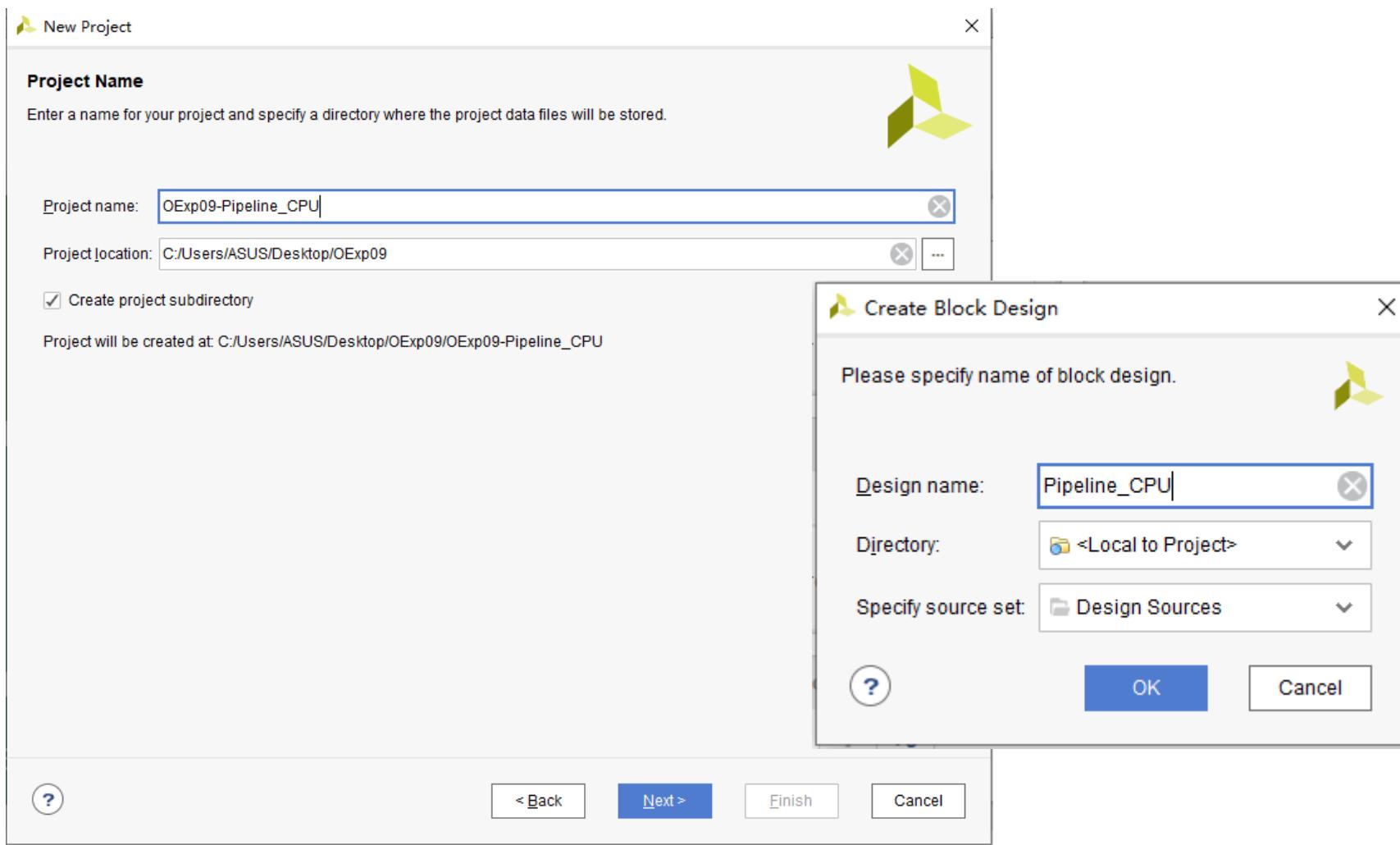
添加模块路径到Write_Back工程目录：

写回模块设计



Ex、Mem、WB模块替换与流水线 CPU集成

流水线CPU集成



清理Exp05-02工程

- 移除工程中的执行、访存、写回模块
- 建议用Exp05-2资源重建工程

设计要点

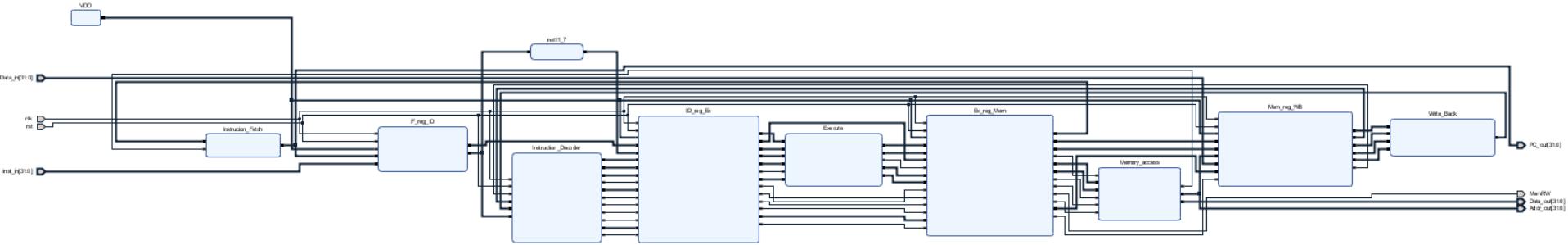
拷贝下列模块到Pipeline_CPU工程目录

本实验设计

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



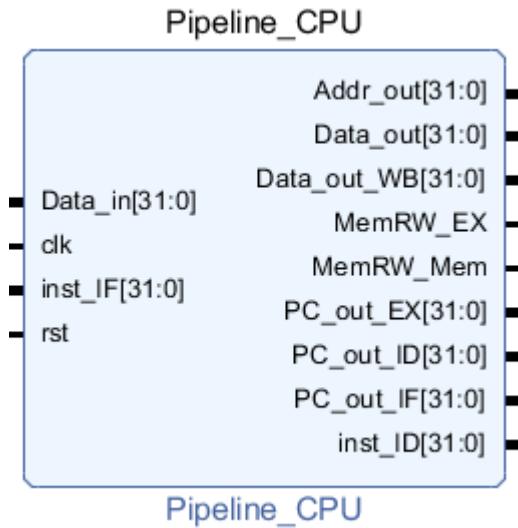
□ 集成Pipeline CPU 的模块层次结构

五级模块、四个寄存器

- Pipeline_CPU (Pipeline_CPU.bd) (1)
 - Pipeline_CPU (Pipeline_CPU.v) (10)
 - > ♦ Ex_reg_Mem : Pipeline_CPU_Ex_reg_Mem_0_0 (Pipeline_CPU_Ex_reg_Mem_0_0.xci)
 - > ♦ Execute : Pipeline_CPU_Pipeline_Ex_0_0 (Pipeline_CPU_Pipeline_Ex_0_0.xci)
 - > ♦ ID_reg_Ex : Pipeline_CPU_ID_reg_Ex_0_0 (Pipeline_CPU_ID_reg_Ex_0_0.xci)
 - > ♦ IF_reg_ID : Pipeline_CPU_IF_reg_ID_0_0 (Pipeline_CPU_IF_reg_ID_0_0.xci)
 - > ♦ Instruction_Decoder : Pipeline_CPU_Pipeline_ID_0_0 (Pipeline_CPU_Pipeline_ID_0_0.xci)
 - > ♦ Instruction_Fetch : Pipeline_CPU_Pipeline_IF_0_0 (Pipeline_CPU_Pipeline_IF_0_0.xci)
 - > ♦ Mem_reg_WB : Pipeline_CPU_Mem_reg_WB_0_0 (Pipeline_CPU_Mem_reg_WB_0_0.xci)
 - > ♦ Memory_Access : Pipeline_CPU_Pipeline_Mem_0_0 (Pipeline_CPU_Pipeline_Mem_0_0.xci)
 - > ♦ Write_Back : Pipeline_CPU_Pipeline_WB_0_0 (Pipeline_CPU_Pipeline_WB_0_0.xci)
 - > ♦ b1 : Pipeline_CPU_xlconstant_0_0 (Pipeline_CPU_xlconstant_0_0.xci)

流水线CPU集成

□ 集成完CPU请参照lab04先完成仿真确认再进行SOC集成



■ 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

The screenshot shows a terminal window displaying the state of a CPU pipeline across five stages: If, Id, Ex, Ma, and Wb. Red arrows point from the right side to specific fields in each stage to identify them.

- If Stage:** Fields include c: 00000000 inst: 00000000, followed by multiple sets of registers (ra, sp, gp, tp) and immediates (t1, t2, s0, s1, a1, a2, a3, a4, a5, a6, a7, s2, s3, s4, s5, s6, s7, s8) with their respective values.
- Id Stage:** Fields include c: 00000000 inst: 00000000, followed by multiple sets of registers (ra, sp, gp, tp) and immediates (t1, t2, s0, s1, a1, a2, a3, a4, a5, a6, a7, s2, s3, s4, s5, s6, s7, s8) with their respective values.
- Ex Stage:** Fields include c: 00000000 inst: 00000000, followed by control signals (d: 00 rs1: 00 rs2: 00 rs1_val: 00000000 rs2_val: 00000000 reg_wen: 0), immediates (s_imm: 0 imm: 00000000 forward_rs1: 00000000 forward_rs2: 00000000), and other status (em_wen: 0 mem_ren: 0 is_branch: 0 is_jal: 0 is_jalr: 0 s_auipc: 0 is_lui: 0 alu_ctrl: 0 cmp_ctrl: 0).
- Ma Stage:** Fields include c: 00000000 inst: 00000000, followed by control signals (d: 00 reg_wen: 0 mem_i_data: 00000000 alu_res: 00000000), and other status (em_wen: 0 mem_ren: 0 is_jal: 0 is_jalr: 0).
- Wb Stage:** Fields include c: 00000000 inst: 00000000, followed by control signals (d: 00 reg_wen: 0 reg_i_data: 00000000).

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
srl 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= 0xFFFFFFFFFB
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)=0x80000000
          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
          srl 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
```

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**