

## Lab05-2

# 流水线处理器—IF、ID设计与集成

Ma De (马德)

[made@zju.edu.cn](mailto:made@zju.edu.cn)

2020

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. 理解流水线取指、译码的设计原理
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. Sword2.0/Sword4.0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，了解取指、译码的原理，掌握IP核的使用方法，集成并测试CPU
- **任务一**：设计取指（IF）、译码（ID）模块，替换实验九的流水线CPU并完成集成
  - ▣ 设计取指模块，替换OExp05-1的取指模块并完成集成
  - ▣ 设计译码模块，替换OExp05-1的译码模块并完成集成
- **任务二**：设计流水线测试方案并完成测试

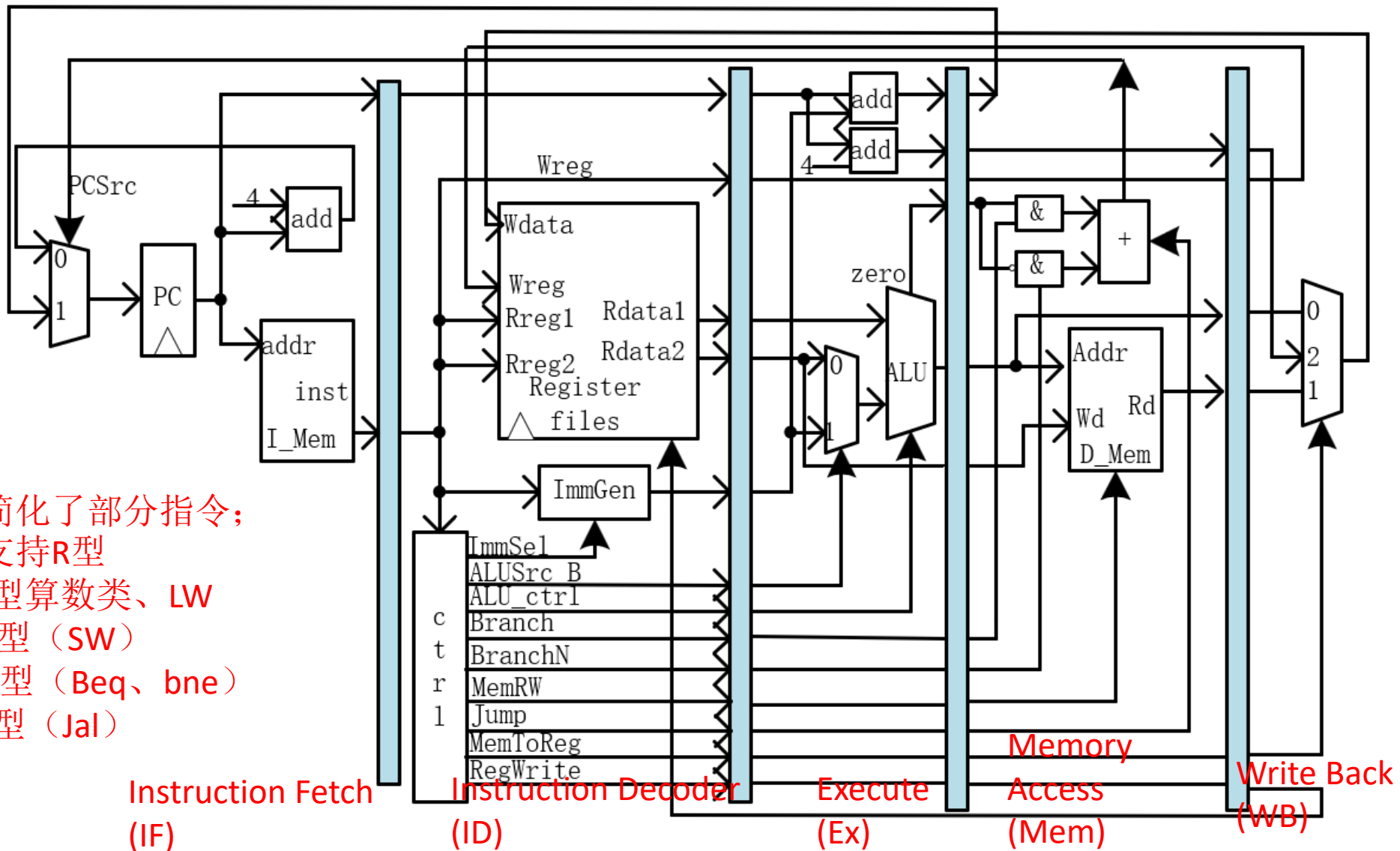
---

# RISC-V 流水线处理器的原理介绍

-----取指（IF）模块介绍

-----译码（ID）模块介绍

# 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)

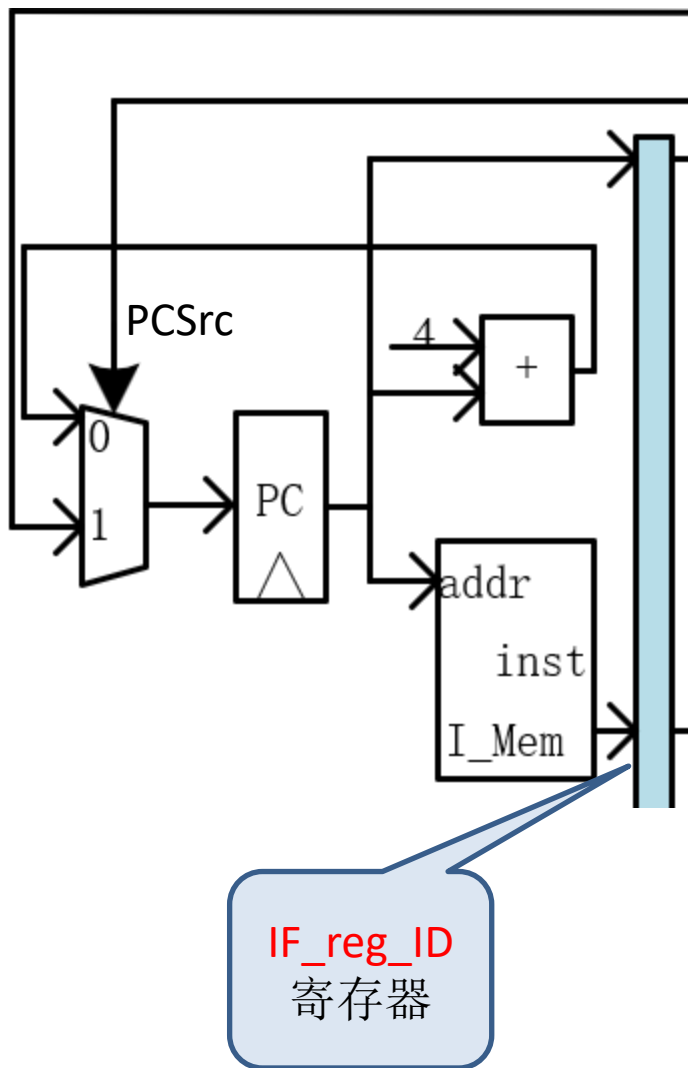
Execute  
(Ex)

Memory  
Access  
(Mem)

Write Back  
(WB)

# 取指—功能介绍

## Instruction Fetch (IF)

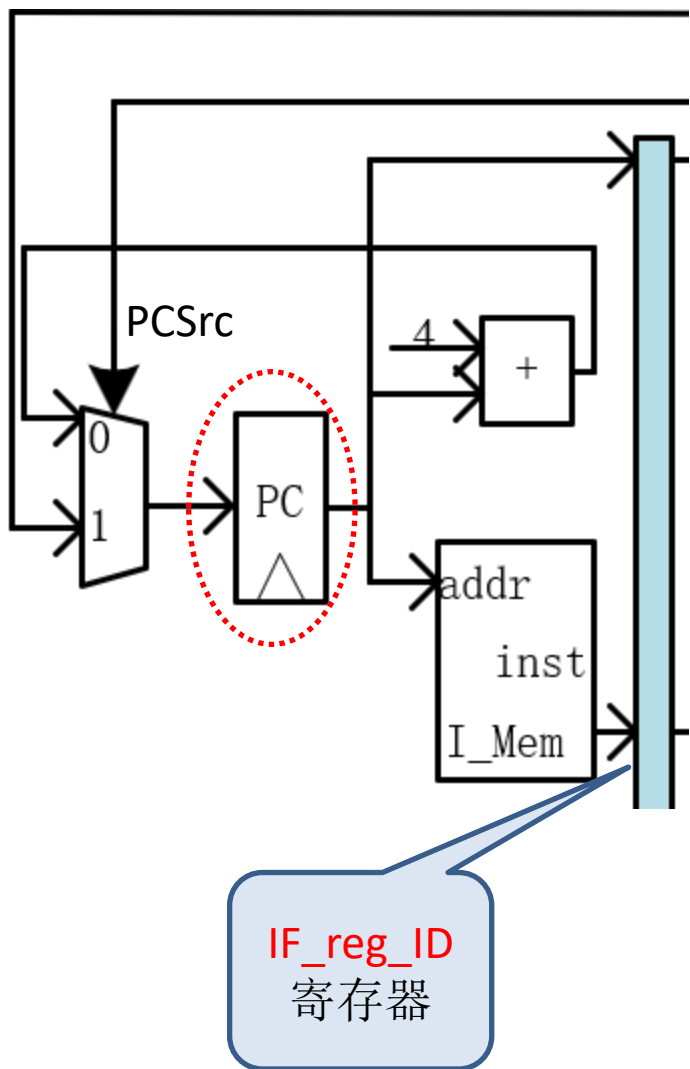


- **取指**：取指阶段涉及程序计数器（PC）和指令存储器（I\_Mem）；程序计数器输出作为地址从指令存储器中读取指令。
- **IF\_reg\_ID**：暂存指令和PC值，以待下一级使用

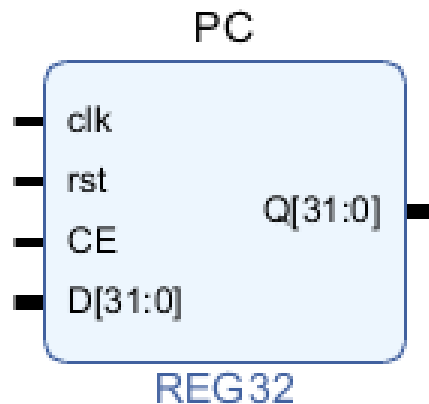


# 取指—部件介绍

## ■ 程序计数器（PC）



- **程序计数器（PC）**：本质是一个32位的寄存器，用于锁存地址，其输出作为访存地址读指令存储器，访存结果为当前指令内容。



# 取指—部件介绍

## ■ 程序计数器（PC）

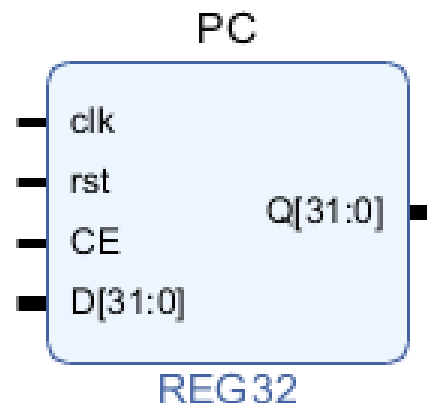
### ■ 模块名：REG32

- 上升沿触发：clk
- 使能信号：CE
- 同步复位：rst=1
- 数据输入：D(31:0)
- 数据输出：Q(31:0)

### ■ 参考描述结构

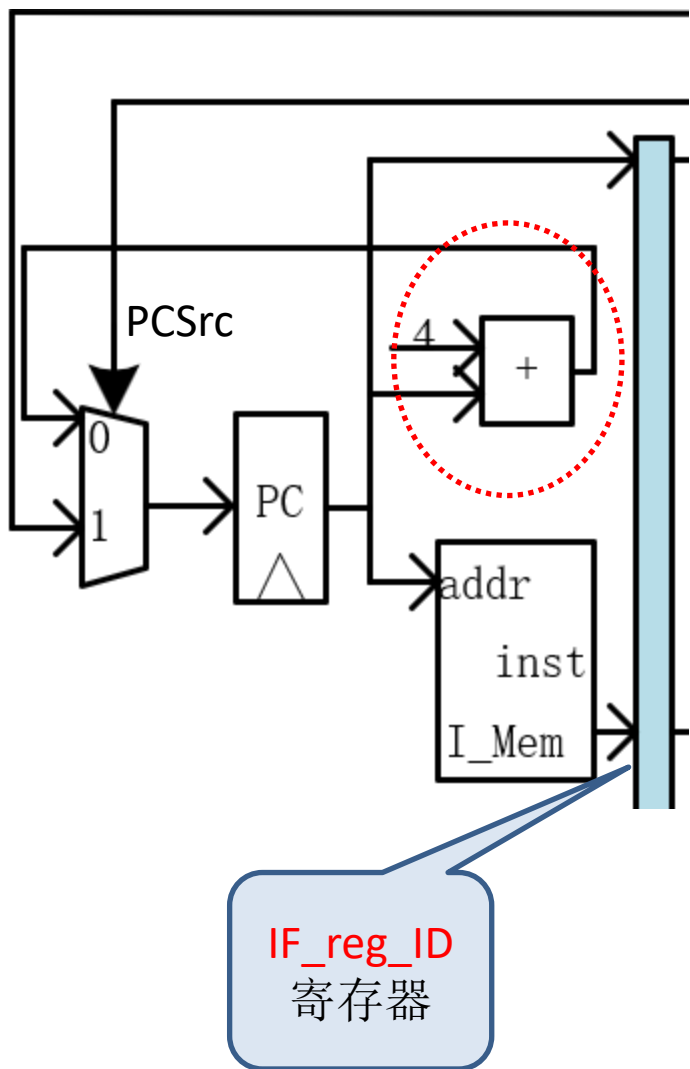
```
module REG32(input clk,  
             .....  
  
             always @(posedge clk or posedge rst)  
                 if (rst==? ) Q <= ? ;  
                 else if (? ) Q <= ? ;  
  
endmodule
```

可直接调用  
OExp04的PC  
寄存器模块

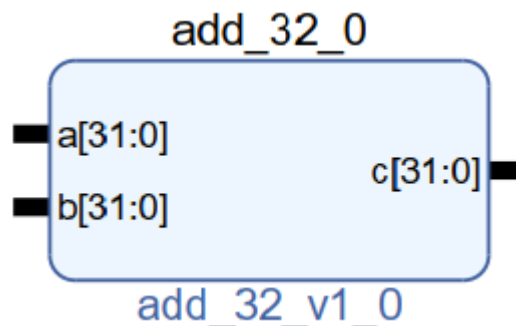


# 取指—部件介绍

## ■ 加法器 (add)



- 加法器 (add)：本质是一个二输入32位的加法器，用于计算PC+4形成顺序执行程序时新的PC值。



# 取指—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)

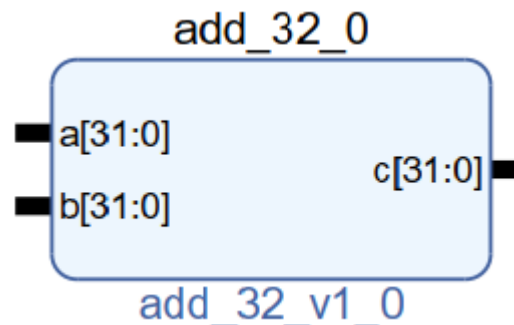
### ■ 参考描述结构

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

.....

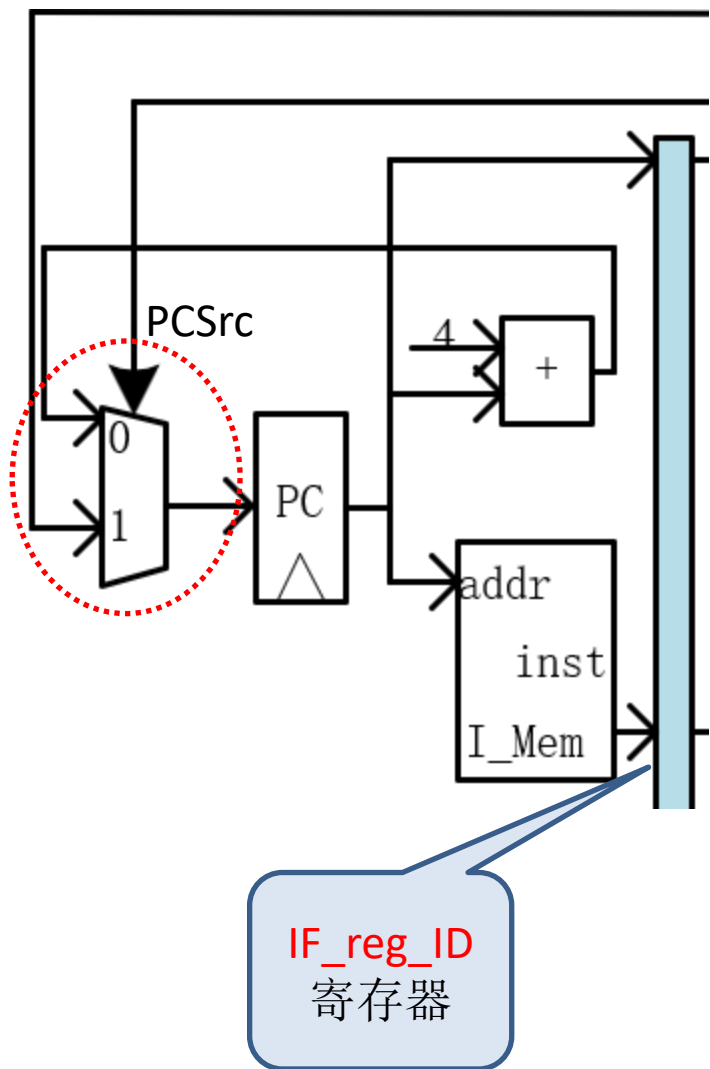
```
endmodule
```

可直接调用  
OExp01的加  
法器模块

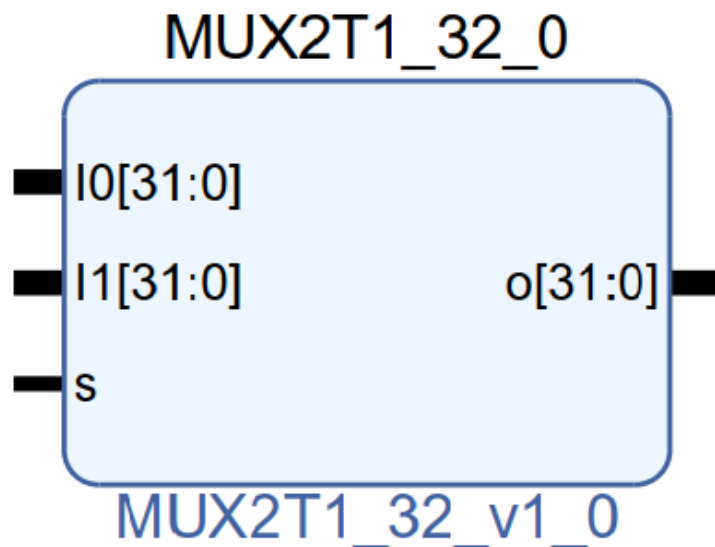


# 取指—部件介绍

## ■ 多选器 (MUX2T1\_32)



- 多选器 (MUX2T1\_32)：本质是一个32位二选一的多路器，用于选择PC的更新值，当PCSrc=0;选择PC+4,当PCSrc=1;选择跳转目标地址。



# 取指—部件介绍

## ■ 多选器 (MUX2T1\_32)

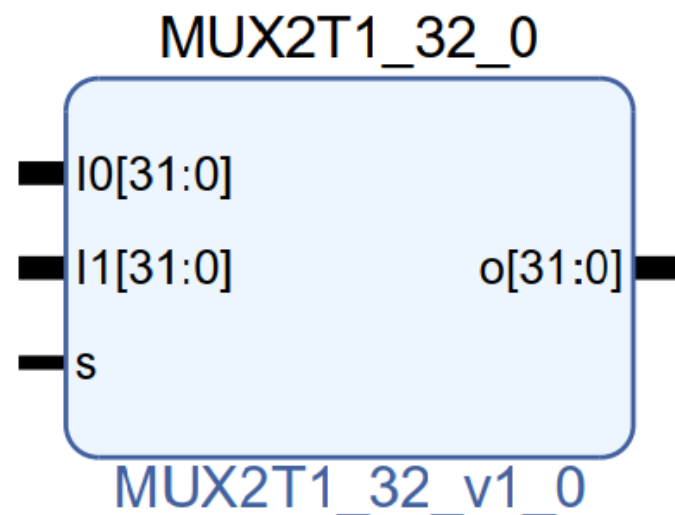
### ■ 模块名: MUX2T1\_32

- 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

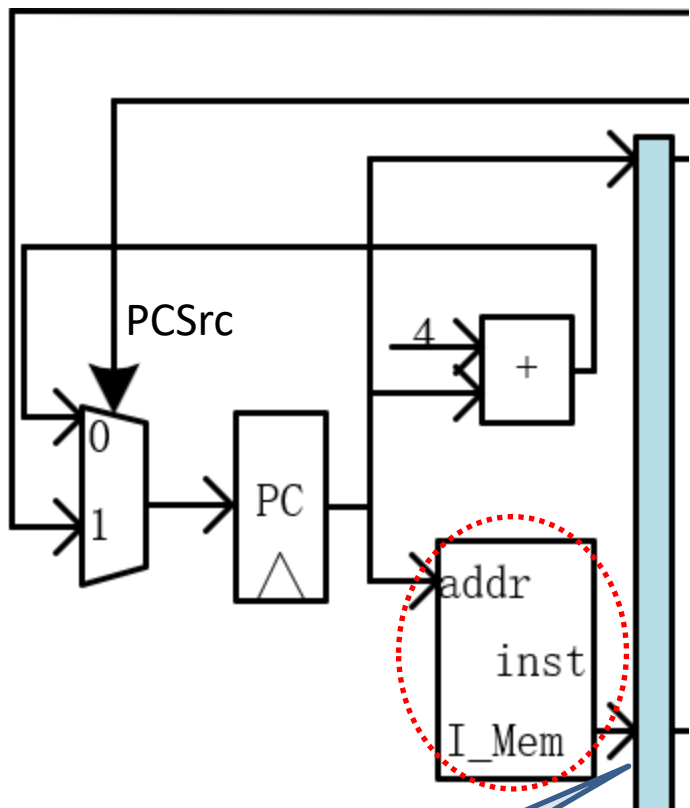
```
module MUX2T1_32(input [31:0] I0,  
                 input [31:0] I1,  
                 input sel,  
                 output reg[31:0] o  
                 );  
    .....  
endmodule
```

可直接调用  
OExp01的多  
选器模块



# 取指—部件介绍

## ■ 指令存储器 (I\_Mem)



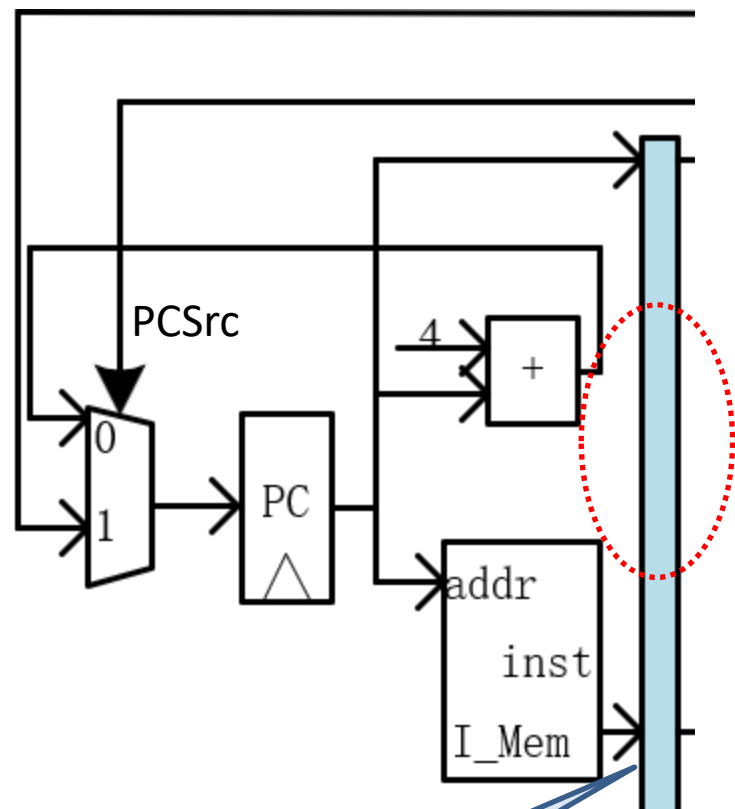
- **指令存储器**：本质是一个异步ROM，输入地址则输出数据，用于存储CPU的指令（**I\_Mem**不是CPU的内部部件，只在构建SOC系统时才用到）。

可直接仿照  
OExp01由IP  
工具生成

IF\_reg\_ID  
寄存器

# 取指—部件介绍

## ■ 取指-译码寄存器 (IF\_reg\_ID)



IF\_reg\_ID  
寄存器

- 取指-译码寄存器 (IF\_reg\_ID) :  
本质是一个寄存器，用于寄存PC值和指令存储器输出的指令。

本实验设计





# 取指—部件介绍

## ■ 取指-译码寄存器 (IF\_reg\_ID)

### ◎ IF\_reg\_ID

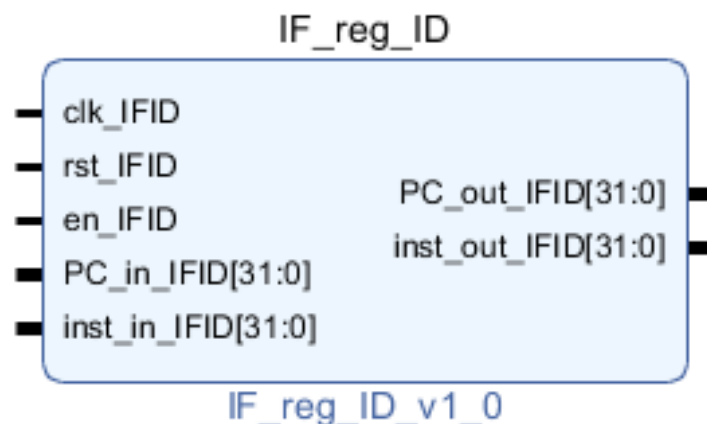
- ☞ 流水线CPU取指和译码之间的寄存器
- ☞ 存储PC值和指令

### ◎ 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

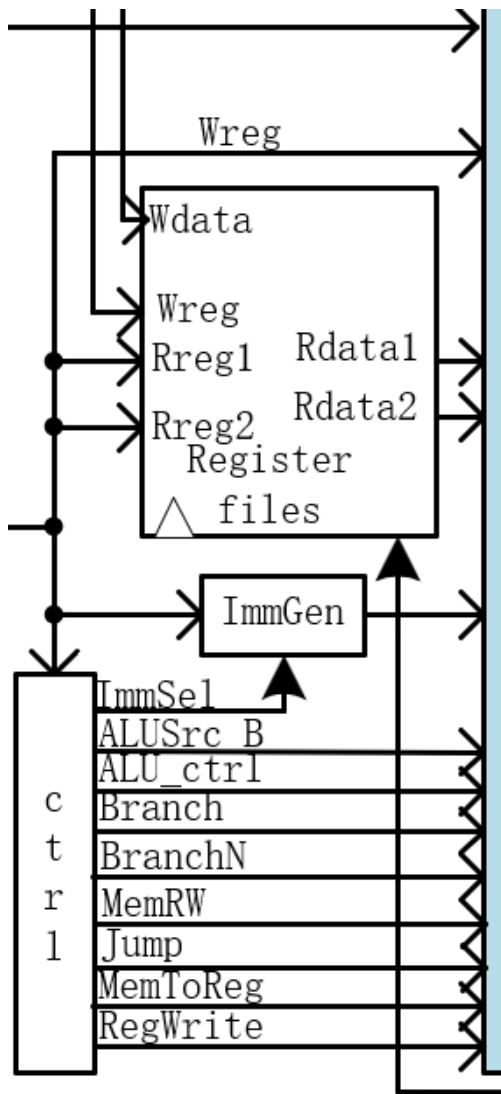
    output reg [31:0] PC_out_IFID,     //PC输出
    output reg [31:0] inst_out_IFID    //指令输出
);
    always @(posedge clk_IFID or posedge rst_IFID)
        if (rst_IFID==? ) ..... <= ? ;
        else if ( ? ) ..... <= ? ;

endmodule
```

# 译码—功能介绍

## ■ Instruction Decoder(ID)

译码器即为单周期CPU中的控制器



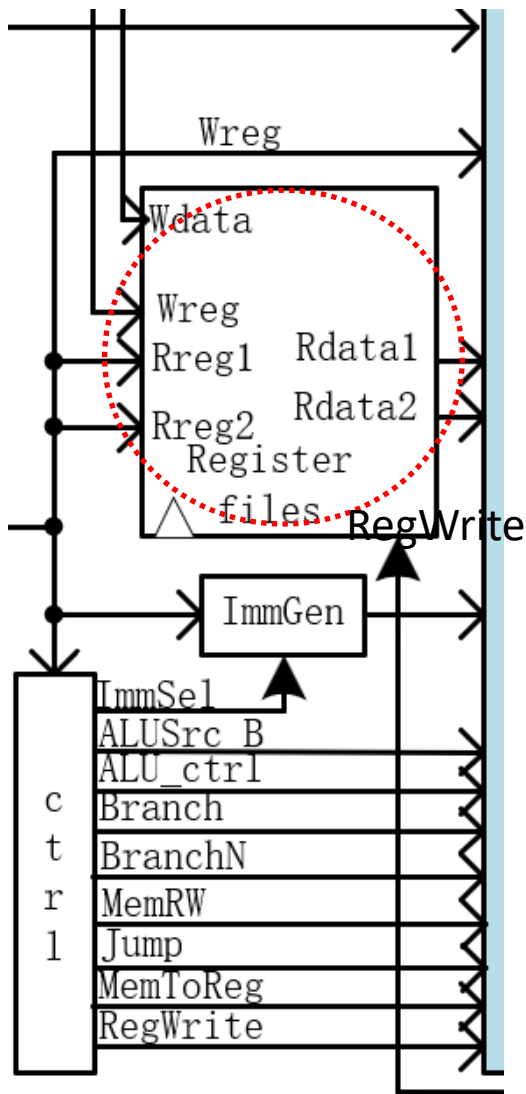
- **译码**：译码阶段涉及寄存器堆（RegisterFiles）和译码器、立即数生成单元（ImmGen）；从寄存器堆可以读取操作数，译码器对指令进行解析产生各种控制信号，立即数生成单元根据控制信号和输入指令生成各种类型的立即数。

- **ID\_reg\_Ex**：暂存PC值，寄存器读取数据，立即数和控制信号以待下一级使用

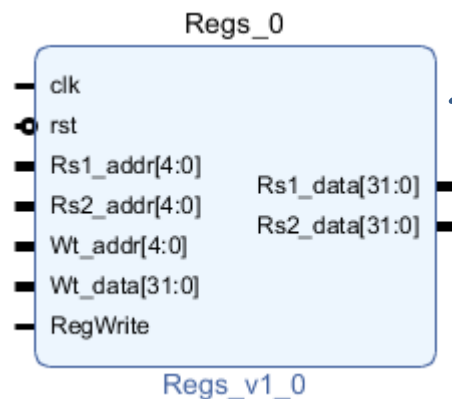
ID\_reg\_Ex  
寄存器

# 译码—部件介绍

## ■ 寄存器堆(RegFiles)



- **寄存器堆**：本质是 $32 \times 32\text{bit}$ 寄存器组，指令操作的主要对象；根据寄存器号读取和写入数据，带有写控制信号。

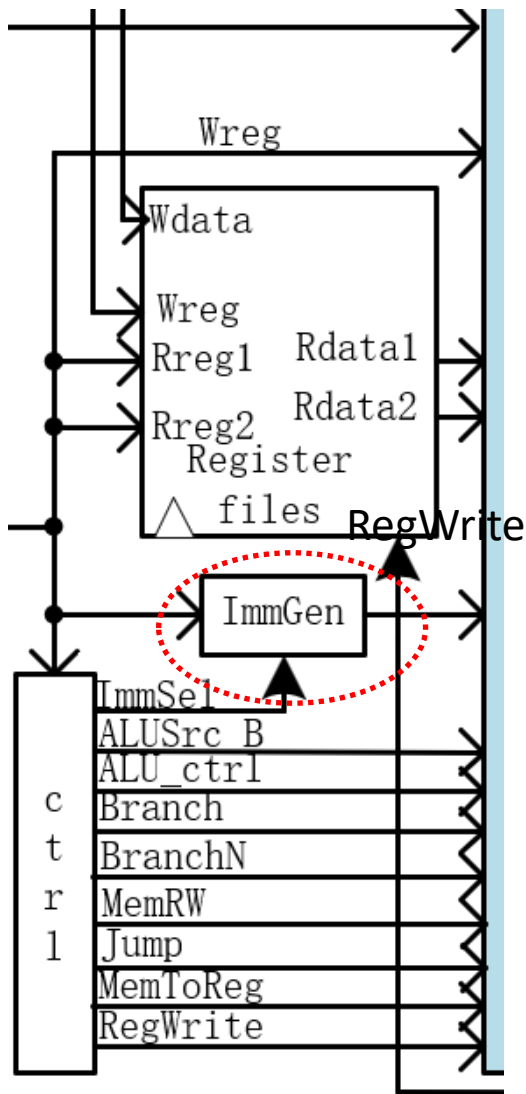


可直接调用  
OExp04寄存器  
堆模块

ID\_reg\_Ex  
寄存器

# 译码—部件介绍

## ■ 立即数生成单元(ImmGen)



- 立即数生成单元：根据输入指令生成立即数。

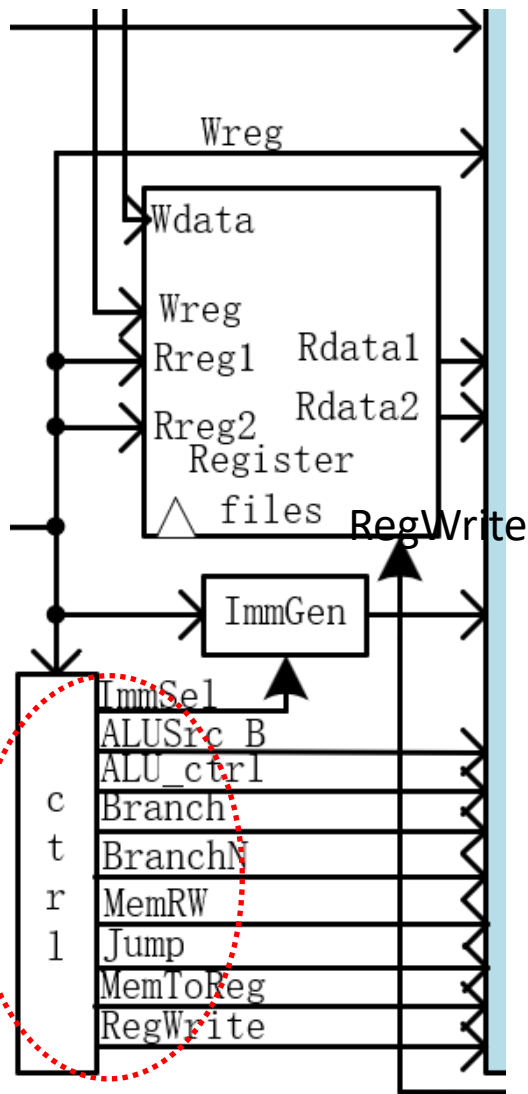


可直接调用  
OExp04立即  
数生成模块

ID\_reg\_Ex  
寄存器

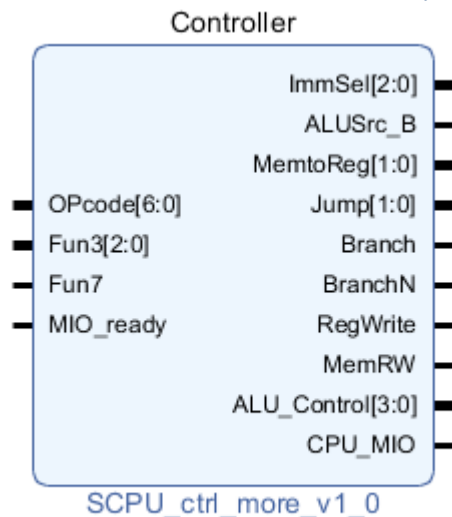
# 译码—部件介绍

## ■ 译码模块/控制器



- **译码模块/控制器**：根据输入指令译码输出各个部件的控制信号。

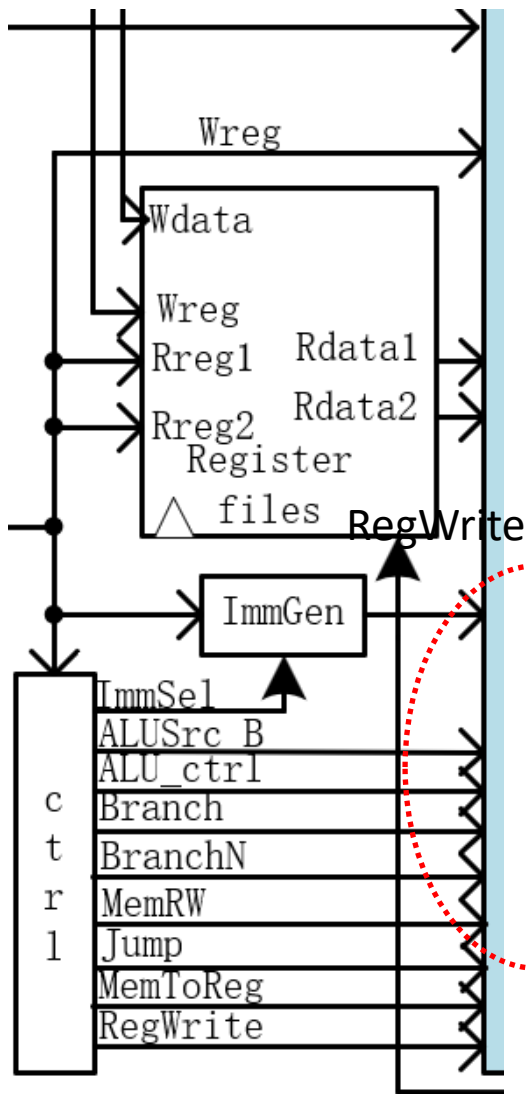
可直接调用  
OExp06控制  
器模块



ID\_reg\_Ex  
寄存器

# 译码—部件介绍

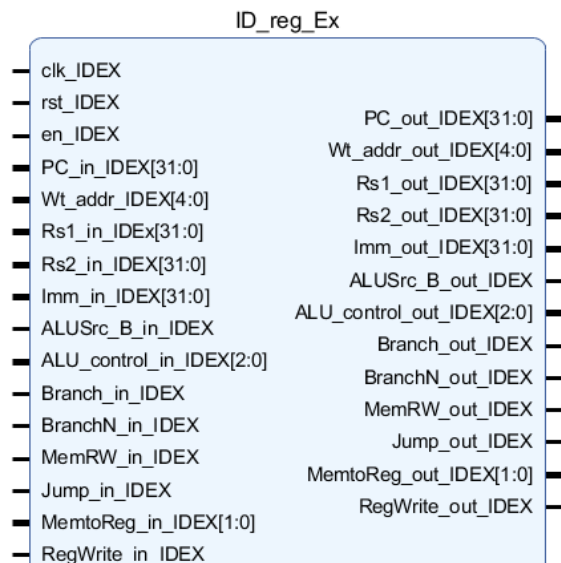
## ■ 译码-执行寄存器 (ID\_reg\_Ex)



■ 译码-执行寄存器：寄存控制信号和ID输出数据。

本实验设计

ID\_reg\_Ex  
寄存器



ID\_reg\_Ex\_v1\_0

- 
- **任务一：**设计取指（IF）、译码（ID）模块，替换实验九的流水线CPU并完成集成
    - ▣ 设计取指模块，替换OExp05-1的取指模块并完成集成
    - ▣ 设计译码模块，替换OExp05-1的译码模块并完成集成



---

# 取指模块、IF\_reg\_ID寄存器 设计集成

## ◎ 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

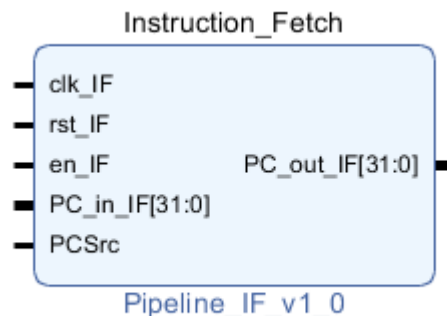
## ◎ 基本功能

- ☞ 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,          //PC输入选择  
    input          rst_IF,          //PC输入选择  
    input          en_IF,           //PC输入选择  
    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  
);  
  
endmodule ( PCSrc=(Branch&zero)|| (BranchN&(~zero))|Jump )
```

# 取指模块设计

 New Project ×

**Project Name**

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


Project name:  ×

Project location:  × ...

☒ Create project subdirectory

Project will be created at: C:/Users/ASUS/Desktop/stall/IP/CPU/pipeline/Pipeline\_IF

? < Back Next > Finish

 Create Block Design ×

Please specify name of block design.

Design name:  ×

Directory:  ▼

Specify source set:  ▼

? OK Cancel

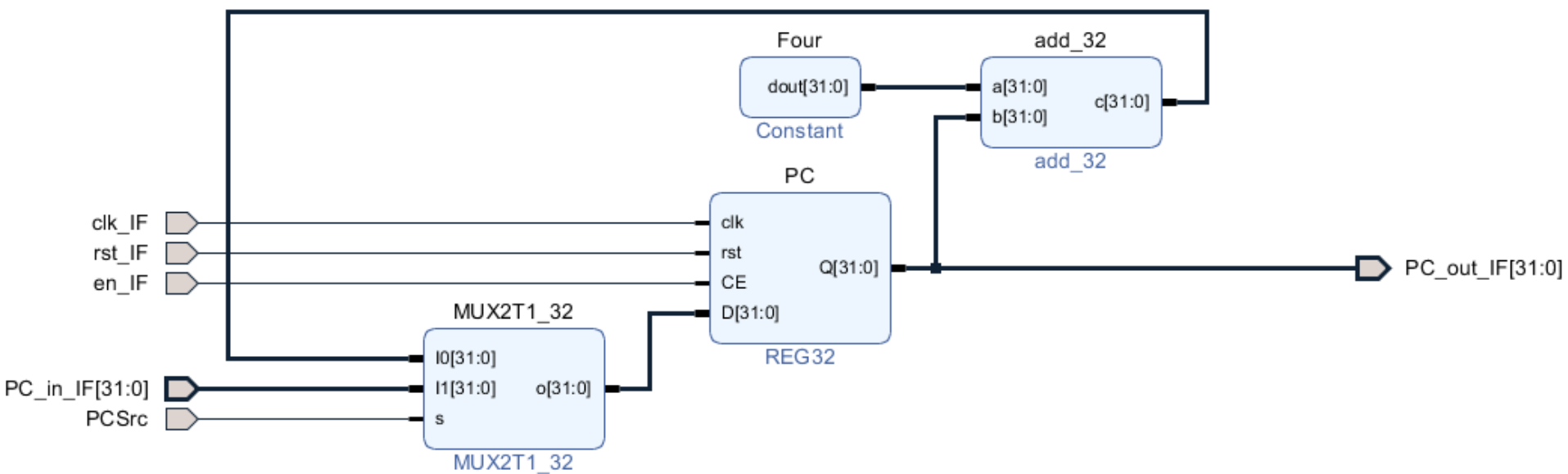
# 取指模块设计

---







拷贝下列模块到Pipeline\_IF工程目录：  
MUX2T1\_32、REG32、add\_32

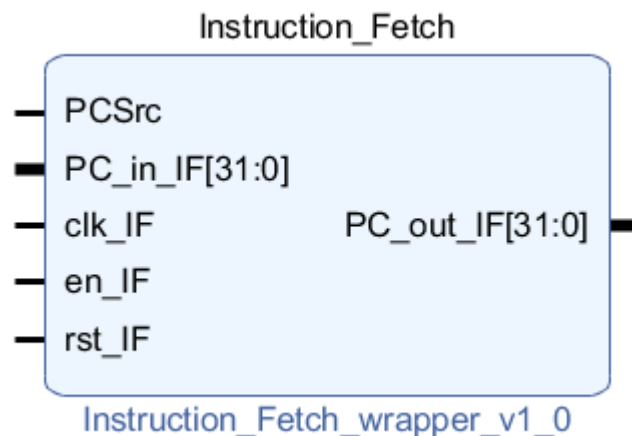
添加模块路径到Pipeline\_IF工程目录：

# 取指模块设计



# 取指模块设计

- ▼  Pipeline\_IF (Pipeline\_IF.bd) (1)
  - ▼  Pipeline\_IF (Pipeline\_IF.v) (4)
    - >  Four : Pipeline\_IF\_xlconstant\_0\_0 (Pipeline\_IF\_xlconstant\_0\_0.xci)
    - >  MUX2T1\_32 : Pipeline\_IF\_MUX2T1\_32\_0\_0 (Pipeline\_IF\_MUX2T1\_32\_0\_0.xci)
    - >  PC : Pipeline\_IF\_REG32\_0\_0 (Pipeline\_IF\_REG32\_0\_0.xci)
    - >  add\_32 : Pipeline\_IF\_add\_32\_0\_0 (Pipeline\_IF\_add\_32\_0\_0.xci)



## ◎ IF\_reg\_ID

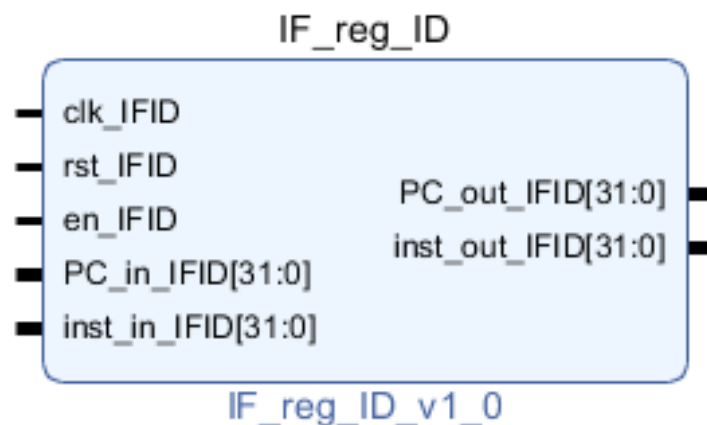
- ☞ 流水线CPU取指和译码之间的寄存器
- ☞ 存储PC值和指令

## ◎ 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
```

---

# 译码模块、ID\_reg\_Ex寄存器 设计集成

## ◎ Pipeline\_ID

☞ 流水线CPU第二阶段

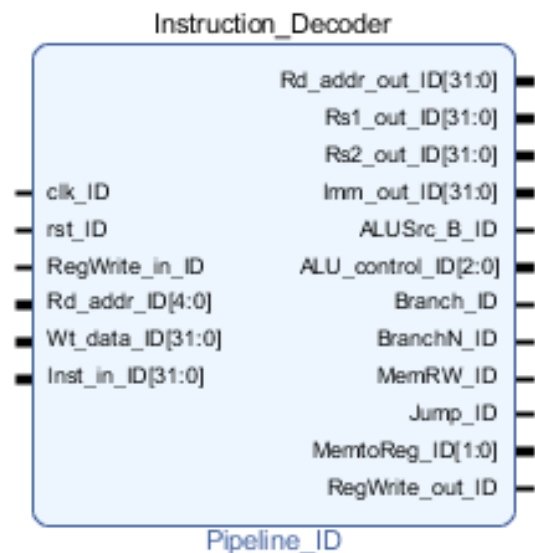
☞ 指令译码

## ◎ 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

---

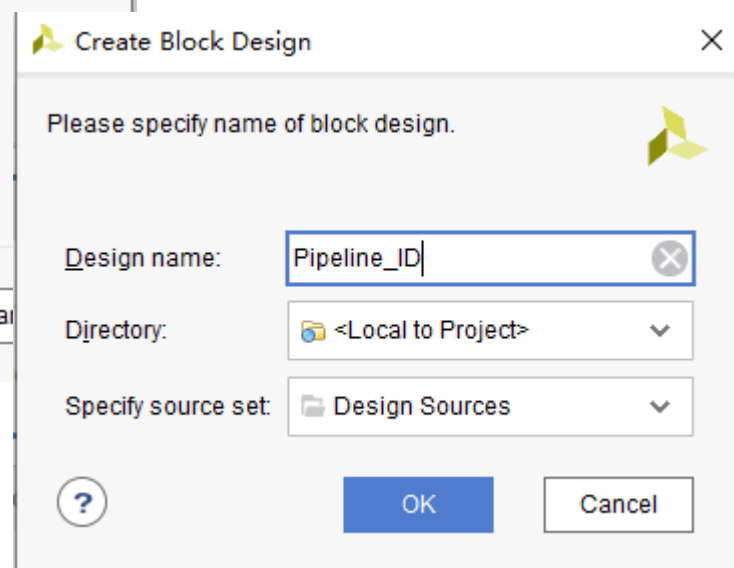
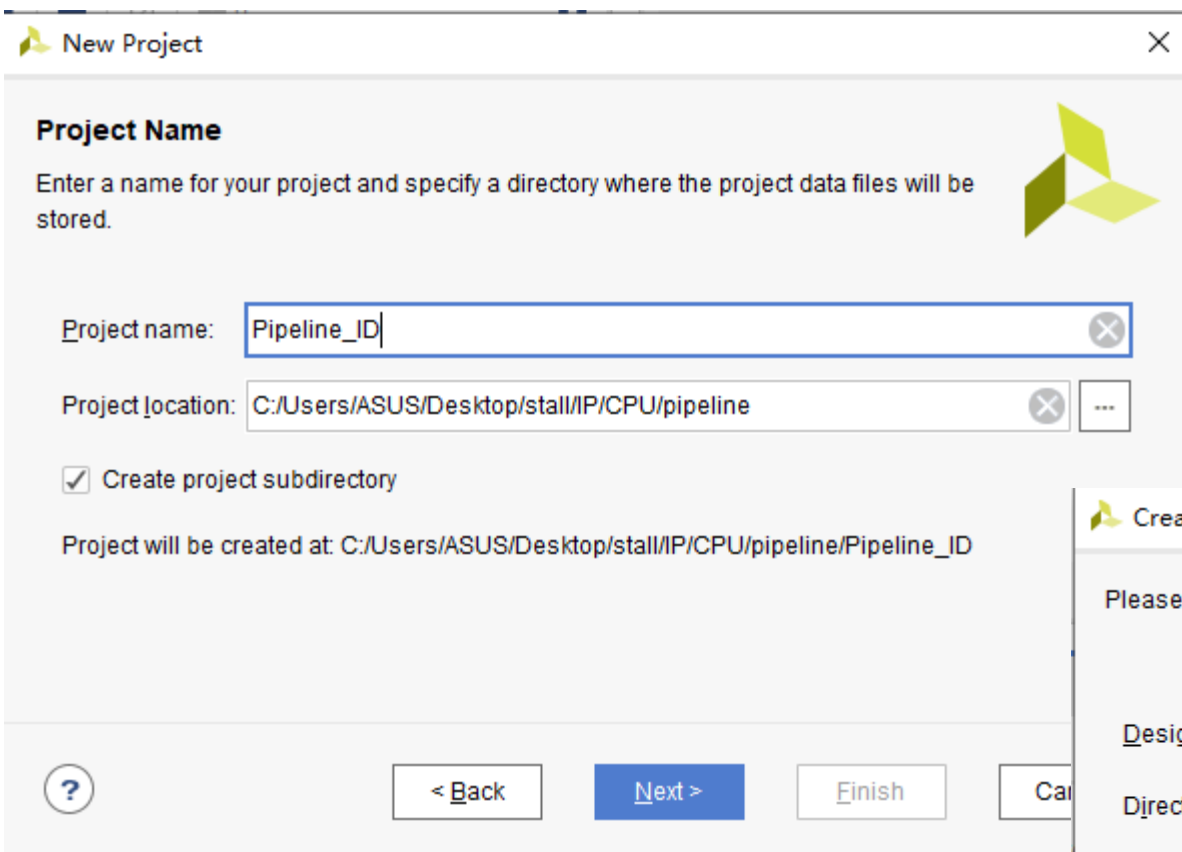
```
module Pipeline_ID(  
    input          clk_ID,           //时钟  
    input          rst_ID,           //复位  
    input          RegWrite_in_ID,   //寄存器堆使能  
    input [4:0]     Rd_addr_ID,       //写目的地址输入  
    input [31:0]    Wt_data_ID,       //写数据输入  
    input [31:0]    Inst_in_ID,       //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

# 译码模块接口： Pipeline\_ID.v

---

```
.....  
output reg [31:0] 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
```

# 译码模块设计



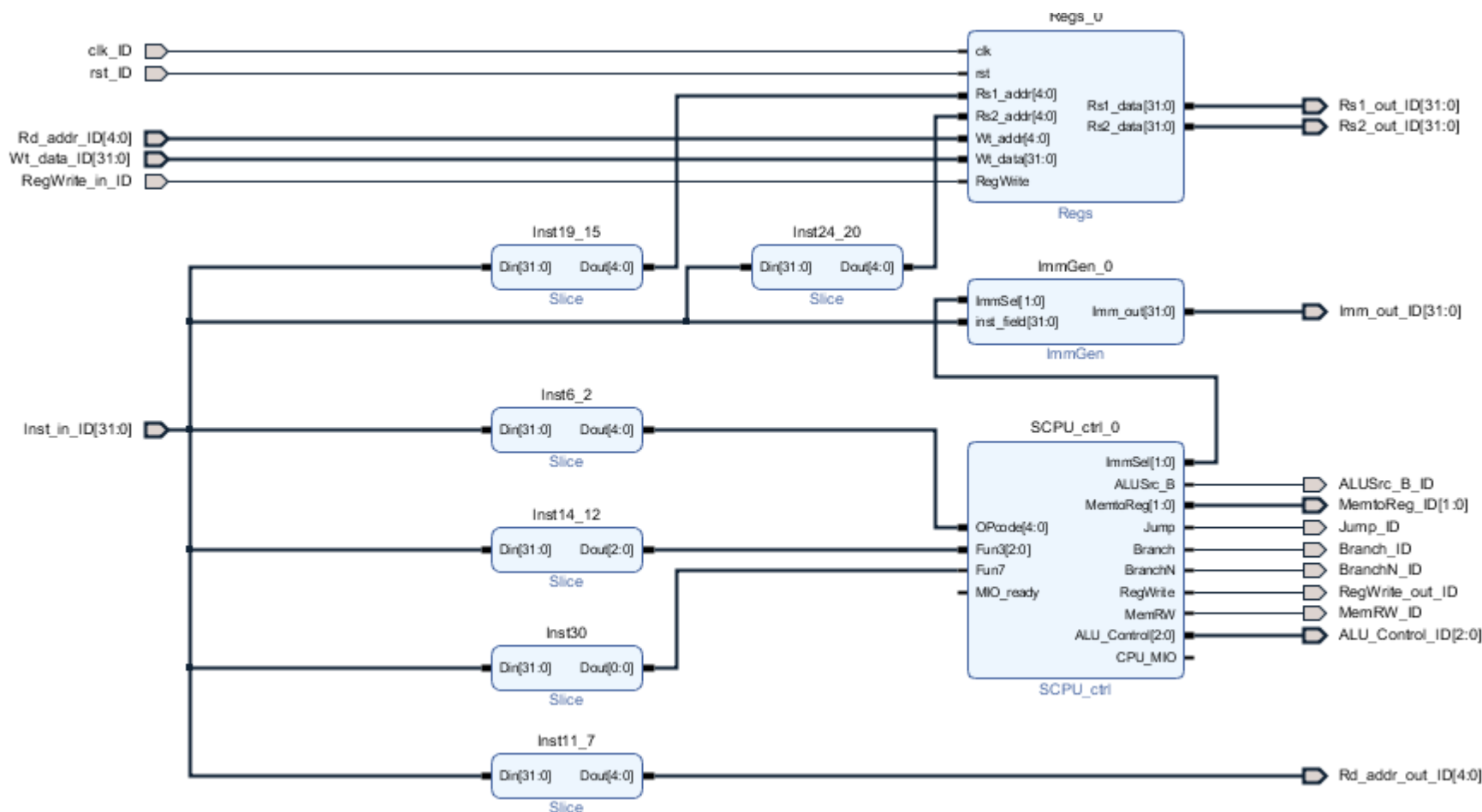
# 译码模块设计

---

**拷贝下列模块到Instruction\_Decoder工程目录：**  
**ImmGen、Regs、SCPU\_Ctrl**

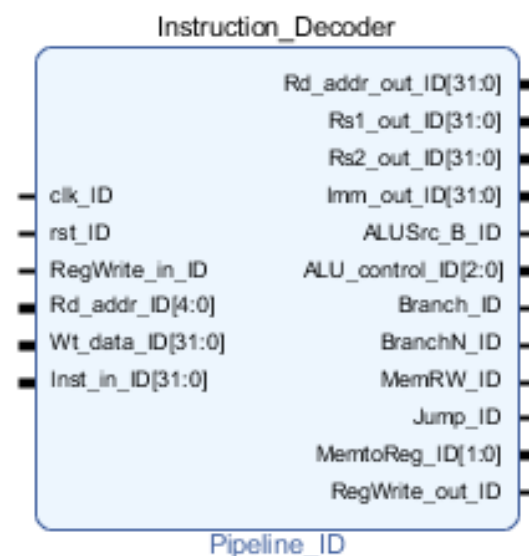
**添加模块路径到Instruction\_Decoder工程目录：**

# 译码模块设计





# 译码模块设计



## ◎ 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_IDEX,        //操作数2输入
    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_IDEX,          //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**

---

# IF、ID模块替换与流水线CPU集成

# 流水线CPU集成

The image shows two overlapping dialog boxes from a software development environment. The background dialog is titled 'New Project' and contains fields for 'Project name' (OExp09-Pipeline\_CPU) and 'Project location' (C:/Users/ASUS/Desktop/OExp09). It also has a checked checkbox for 'Create project subdirectory' and a summary line stating the project will be created at a specific path. The foreground dialog is titled 'Create Block Design' and prompts the user to specify the name of the block design. It has a 'Design name' field (Pipeline\_CPU), a 'Directory' dropdown menu (set to '<Local to Project>'), and a 'Specify source set' dropdown menu (set to 'Design Sources'). Both dialogs have 'OK' and 'Cancel' buttons. The 'New Project' dialog also features a 'Back' button and a 'Next >' button at the bottom.

**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

**Create Block Design**

Please specify name of block design.

Design name: Pipeline\_CPU

Directory: <Local to Project>

Specify source set: Design Sources

OK Cancel

< Back Next > Finish Cancel

# 清理Exp05-1工程

---

- 移除工程中的取指、译码模块
- 建议用Exp05-1资源重建工程

# 设计要点

---

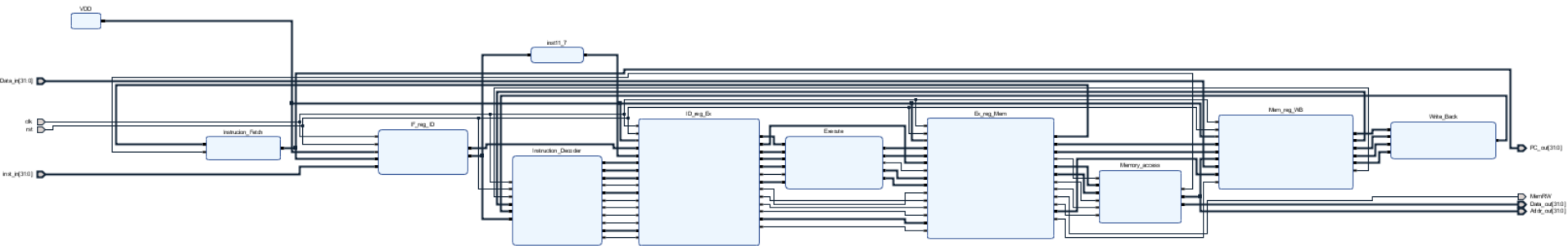
本实验设计

**拷贝下列模块到Pipeline\_CPU工程目录:**  
**Pipeline\_IF、IF\_reg\_ID、Pipeline\_ID、ID\_reg\_Ex、**

**添加模块路径到Pipeline\_CPU工程目录:**

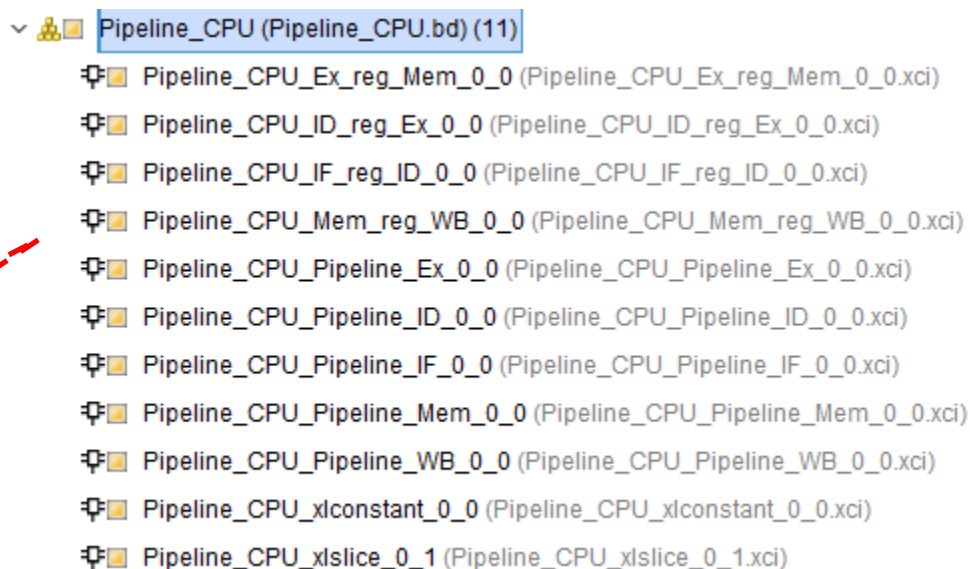


# 流水线CPU集成---替换取指、译码模块

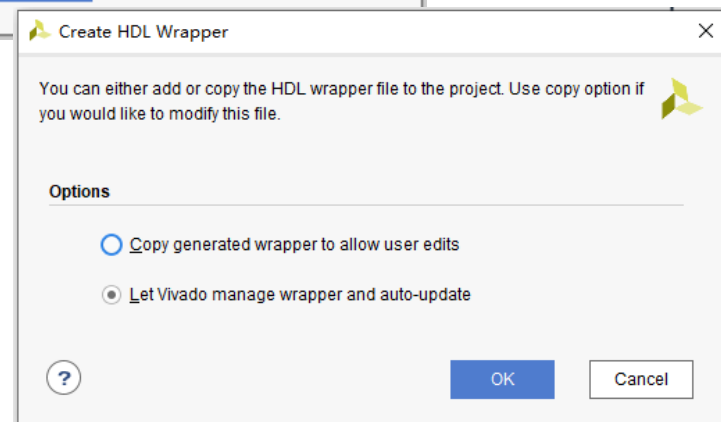
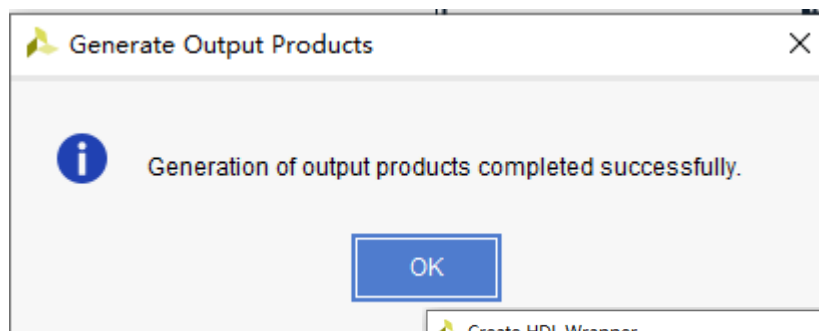
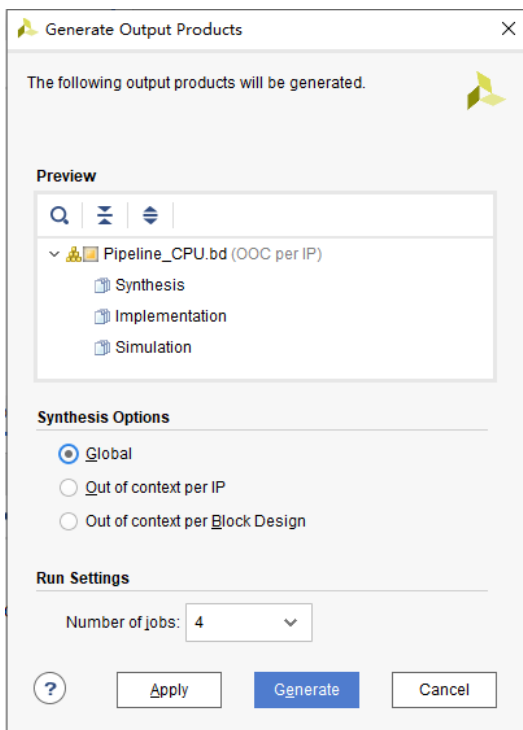
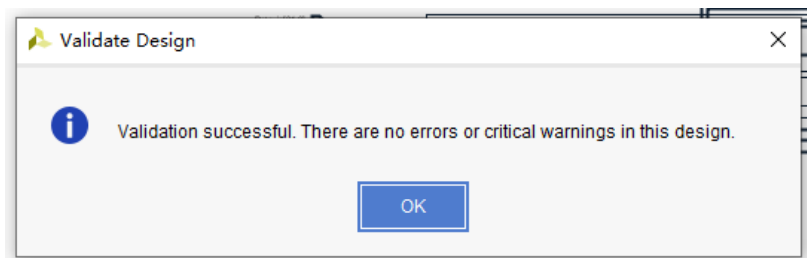


## □ 集成Pipeline CPU 的模块层次结构

五级模块、四  
个寄存器



# 流水线CPU集成



---



---

## ■ 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

```
===== If =====
c: 00000000 inst: 00000000

===== Id =====
c: 00000000 inst: 00000000
0: 00000000 ra: 00000000 sp: 00000000 gp: 00000000 tp: 00000000
0: 00000000 t1: 00000000 t2: 00000000 s0: 00000000 s1: 00000000
0: 00000000 a1: 00000000 a2: 00000000 a3: 00000000 a4: 00000000
5: 00000000 a6: 00000000 a7: 00000000 s2: 00000000 s3: 00000000
4: 00000000 s5: 00000000 s6: 00000000 s7: 00000000 s8: 00000000
9: 00000000 s10: 00000000 s11: 00000000 t3: 00000000 t4: 00000000
5: 00000000 t6: 00000000

===== Ex =====
c: 00000000 inst: 00000000
d: 00 rs1: 00 rs2: 00 rs1_val: 00000000 rs2_val: 00000000 reg_wen: 0
s_inn: 0 inn: 00000000 forward_rs1: 00000000 forward_rs2: 00000000
en_wen: 0 mem_ren: 0 is_branch: 0 is_jal: 0 is_jalr: 0
s_auiopc: 0 is_lui: 0 alu_ctrl: 0 cmp_ctrl: 0

===== Ma =====
c: 00000000 inst: 00000000
d: 00 reg_wen: 0 mem_i_data: 00000000 alu_res: 00000000
en_wen: 0 mem_ren: 0 is_jal: 0 is_jalr: 0

===== Wb =====
c: 00000000 inst: 00000000
d: 00 reg_wen: 0 reg_i_data: 00000000
```

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= 0xFFFFFFFFB
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
      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
```

# 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**