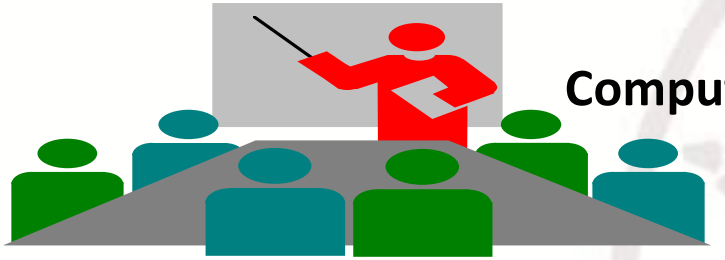


Computer Organization & Design
实验与课程设计

实验十

多周期CPU设计-数据通路设计

施青松
Asso. Prof. Shi Qingsong
College of Computer Science and Technology, Zhejiang University
zjsqs@zju.edu.cn

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验目的

1. 深入运用寄存器传输控制技术
2. 深入掌握CPU的核心：数据通路组成与原理
3. 设计多周期数据通路
4. 测试方案的设计
5. 测试程序的设计

实验环境

实验设备

1. 计算机（Intel Core i5以上，4GB内存以上）系统

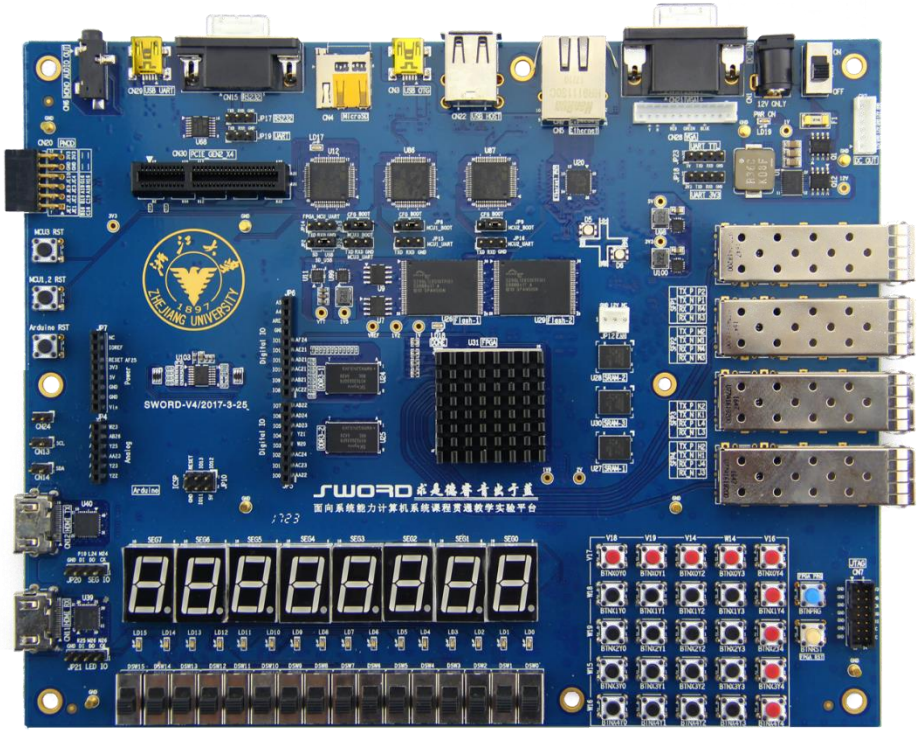
2. 计算机软硬件课程贯通教学实验系统(SWORD4.0)

3. Xilinx ISE14.7及以上开发工具

材料

无

计算机软硬件课程贯通教学实验系统



贯通教学实验平台主要参数

▼ **核心芯片**

Xilinx Kintex™-7系列的XC7K325资源：

162,240个，Slice：25350，片内存储：11.7Mb

▼ **存储体系** 支持32位存储层次体系结构

6MB SRAM静态存储器：支持32Data，16位TAG

512M BDDR3动态存储：支持32Data

32MB NOR Flash存储 ：支持32位Data

▼ **基本接口** 支持微机原理、SOC或微处理器简单应用

4×5+1矩阵按键、16位滑动开关、16位LED、8位七段数码管

▼ **标准接口** 支持基本计算机系统实现

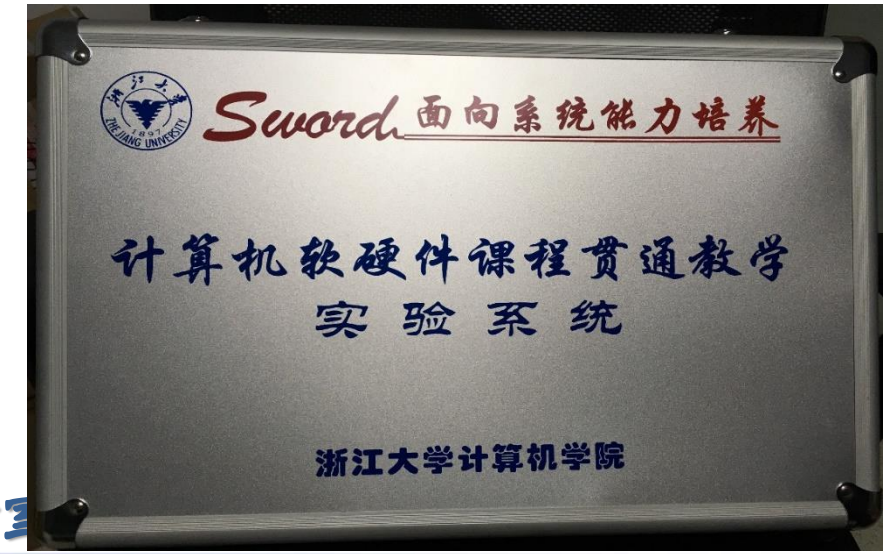
12位VGA接口（RGB656）、USB-HID（键盘）

▼ **通讯接口** 支持数据传输、调试和网络

UART接口、 10M/100M/1000M以太网、 SFP光纤接口

▼ **扩展接口** 支持外存、多媒体和个性化设备

MicroSD(TF）、PMOD、 HDMI、Arduino



Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验任务

1. 设计9+条指令的多周期数据通路

- 设计多周期数据通路逻辑原理图

  R-Type：add，sub，and，or，slt，nor*；

  I-Type： lw，sw， beq；

  J-Type： J

- 用硬件描述语言设计实现数据通路
  - ALU和Regs调用Exp04设计的模块
- 替换Exp09数据通路核

2. 数据通路测试

- 设计测试方案与测试程序
- 通路测试：I-格式通路、R-格式通路

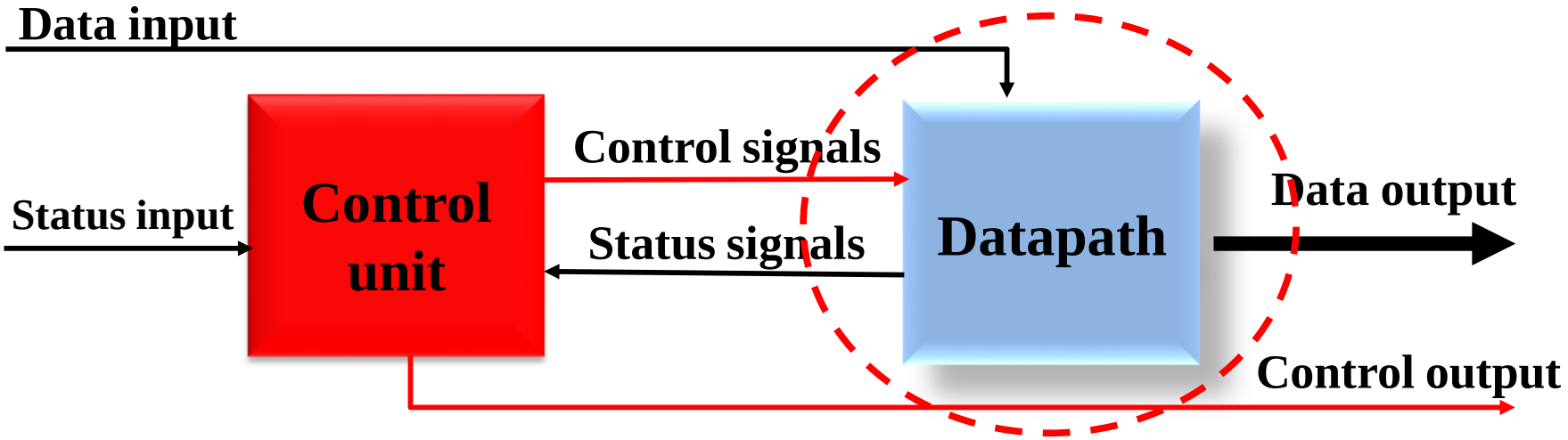
Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

CPU organization

Digital circuit

- General circuits that controls logical event with logical gates --Hardware



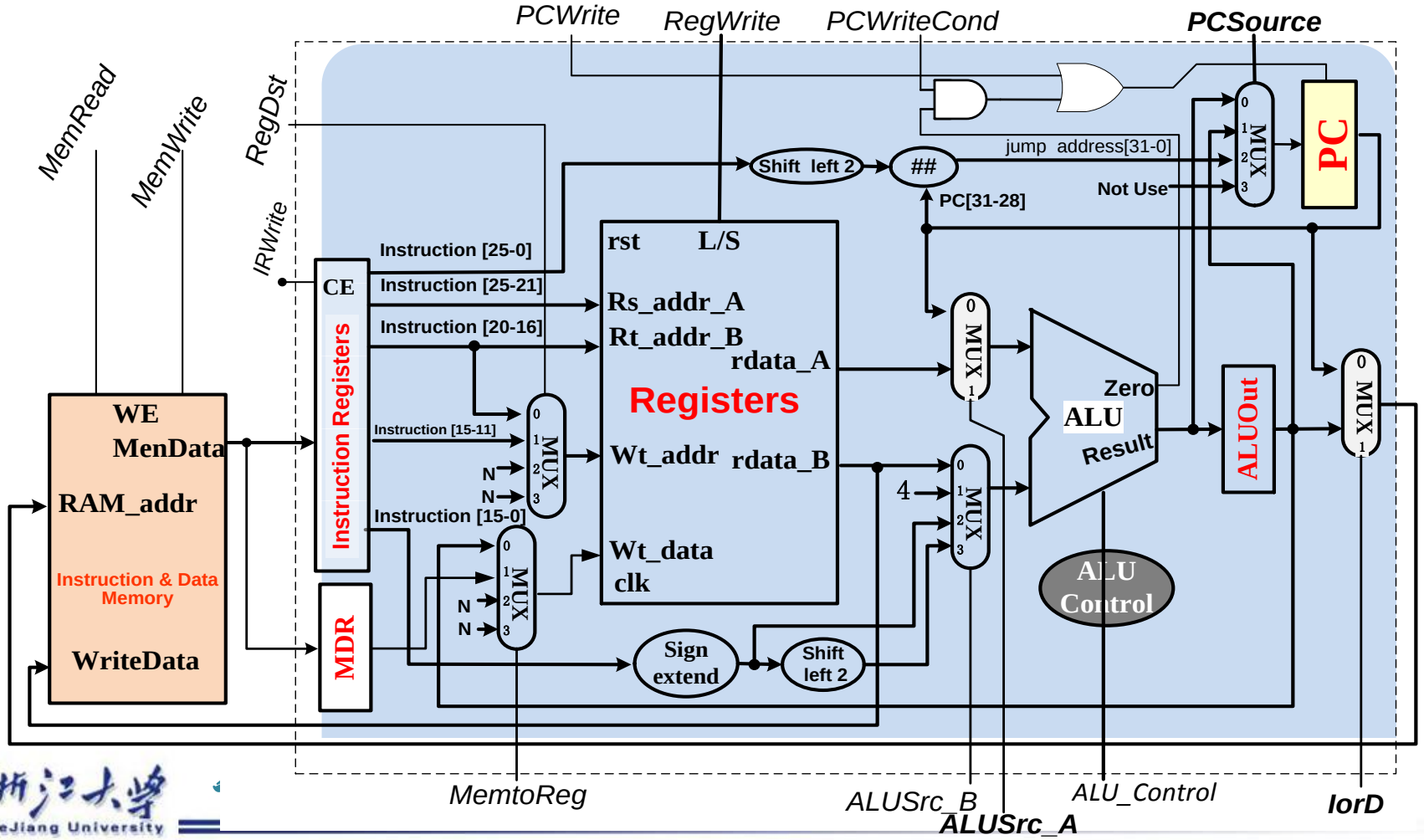
Computer organization

- Special circuits that processes logical action with instructions -Software

多周期数据通路结构：兼容9-23+指令

MUX选择更多输入以兼容扩展

- 找出指令的通路：5+1个MUX
  - 比单周期增加了什么通道？



多周期控制信号定义： Defined 10+6+？ control

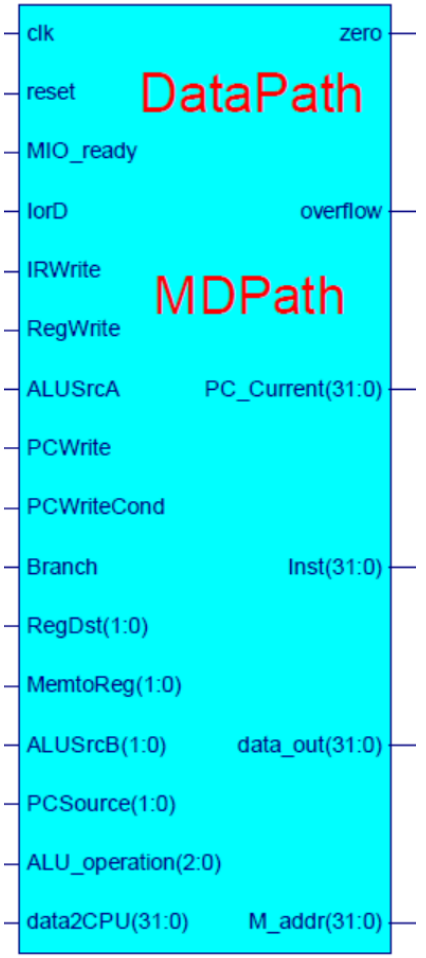
- 修改单周期通路与操作控制

信号	源数目	功能定义	赋值0时动作	赋值1时动作
ALUScrA ALUSrc_B	?	ALU端口A、B输入选择		
RegDst	?	寄存器写地址选择(考虑扩展)		
MemtoReg	?	寄存器写数据选择(考虑扩展)		
IorD	?	新增		
PCSource	?	新增		
PCWriteCond	?	新增		
……	?	新增		
Branch	?	Beq指示(考虑Bne扩展)		
RegWrite	-	寄存器写控制		
MemWrite	-	存储器写控制		
MemRead	-	存储器读控制		
ALU_Control	000- 111	3位ALU操作控制	参考表 Exp04	Exp04

请填写信号赋值时对应操作

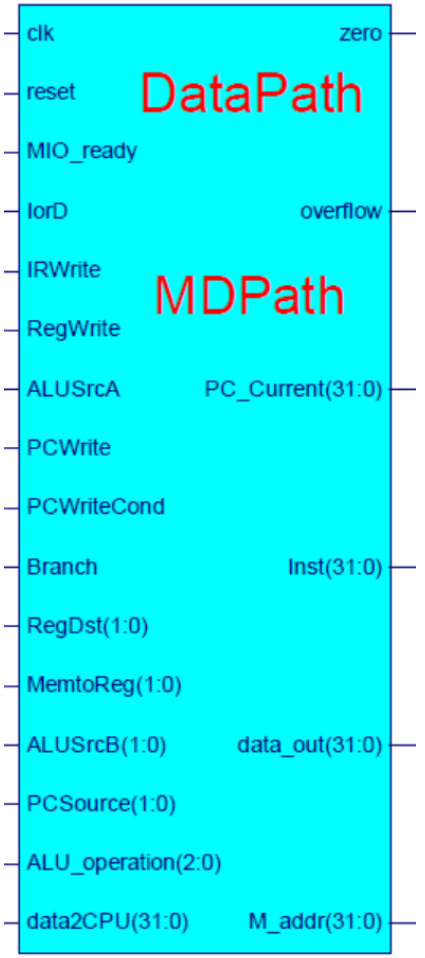
多周期数据通路模块： MDPath

- 数据通路
  - CPU主要部件之一
  - 寄存器传输控制对象：通用数据通路
- 基本功能
  - 具有通用计算功能的算术逻辑部件
  - 具有通用目的寄存器
  - 具有通用计数所需的尽可能的路径
- 重要信号
  - Inst_R：指令寄存器输出
  - PC_Current：当前PC(PC+4)
  - M_addr：存储器地址
  - Branch：=1→beq；=0 → bne
  - PCWriteCond：Branch指令



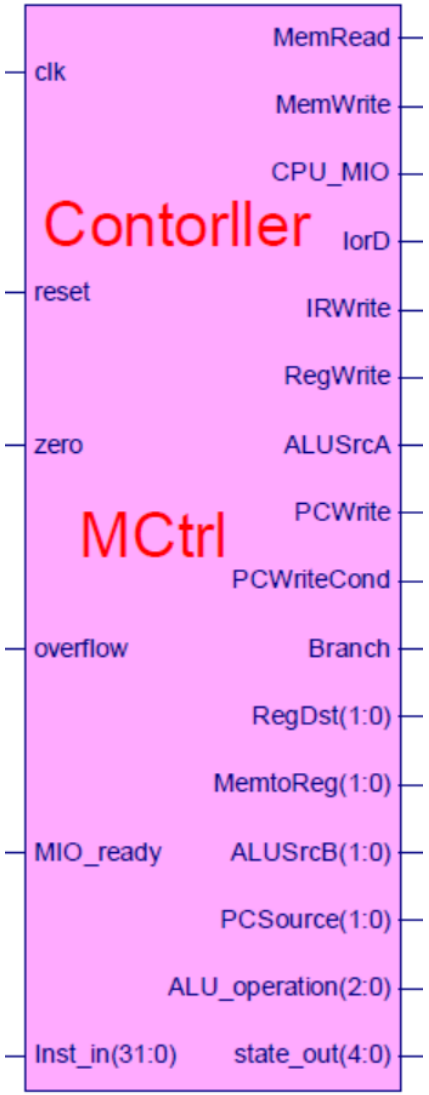
数据通路接口参考- MDPath.v

```
module      MDPath(input clk,
                   input reset,
                   input MIO_ready,              //外部输入=1
                   input IorD,
                   input IRWrite,
                   input[1:0] RegDst,            //预留到2位
                   input RegWrite,
                   input[1:0]MemtoReg,           //预留到2位
                   input ALUSrcA,
                   input[1:0]ALUSrcB,
                   input[1:0]PCSource,           //4选1控制
                   input PCWrite,
                   input PCWriteCond,
                   input Branch,
                   input[2:0]ALU_operation,

                  output[31:0]PC_Current,
                  input[31:0]data2CPU,
                  output[31:0]Inst,
                  output[31:0]data_out,
                  output[31:0]M_addr,
                  output zero,
                  output overflow
                  );
endmodule
```



CPU部件之二-控制器：MCtrl

- 本实验用IP固核- **MCtrl**
  - 核调用模块ctrl.ngc
  - 核接口信号模块(空文档)：ctrl.v
  - 核模块符号文档：ctrl.sym
- 重要信号
  - MIO_ready：外设就绪
    - =0 CPU等待
    - =1 CPU正常运行
    - 本实验恒等于1
  - Inst_in：指令输入，来自IR输出
  - State_out：状态编码，用于测试



控制器接口文档- MCtrl.v

```
module   MCtrl(input  clk,
               input  reset,
               input  [31:0] Inst_in,
               input  zero,
               input  overflow,
               input  MIO_ready,                          //外部输入=1
               output reg MemRead,
               output reg MemWrite,
               output reg[2:0]ALU_operation,              //ALU_Control
               output [4:0]state_out,

               output reg CPU_MIO,
               output reg IorD,
               output reg IRWrite,
               output reg [1:0]RegDst,                    //预留到2位
               output reg RegWrite,
               output reg [1:0]MemtoReg,                  //预留到2位
               output reg ALUSrcA,
               output reg [1:0]ALUSrcB,
               output reg [1:0]PCSource,
               output reg PCWrite,
               output reg PCWriteCond,
               output reg Branch
               );
endmodule
```

U3-存储器初始化数据参考文档：mem.coe 代码与数据共存

memory_initialization_radix=16;　　9条指令设计的，根据实验三修改

memory_initialization_vector=

```
00A52820, AC650000, 8C650000, 00A85824, 01A26820, 11A00017, 8C650000, 01CE9020, 0252B020, 02569020,
00B25824, 11600005, 1172000A, 01CE9020, 1172000B, AC890000, 08000036, 11410001, 0800004D, 00005027,
014A5020, AC8A0000, 08000036, 8E290860, AC890000, 08000036, 8E290820, AC890000, 08000036, 8C0D0014,
014A5020, 01425025, 022E8820, 02348824, 01224820, 11210001, 0800005F, 000E4820, 01224820, 8C650000,
00A55820, 016B5820, AC6B0000, AC660004, 0800003E, 00000000, 00000000, 00000000, 00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,
00000000, 00000000,    ……
……
```

代码区：地址从00000000开始

```
F0000000, 000002AB, 80000000, 0000003F, 00000001, FFFF0000, 0000FFFF, 80000000, 00000000, 11111111,
22222222, 33333333, 44444444, 55555555, 66666666, 77777777, 88888888, 99999999, AAAAAAAA, BBBBBBBB,
CCCCCCCC, DDDDDDDD, EEEEEEEE, FFFFFFFF, 557EF7E0, D7BDFBD9, D7DBFDB9, DFCFFCFB, DFCFBFFF, F7F3DFFF,
FFFFDF3D, FFFF9DB9, FFFFBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF,
03BDF020, 03DEF820, 08002300, 00000000, 00000000, 00000000, 00000000, 00000000;
```

数据区：地址起始需要约定：此代码为00000200

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

CPU之数据通路设计

-用HDL描述实现

-调用实验一设计的多路器

-调用实验一的基本运算模块

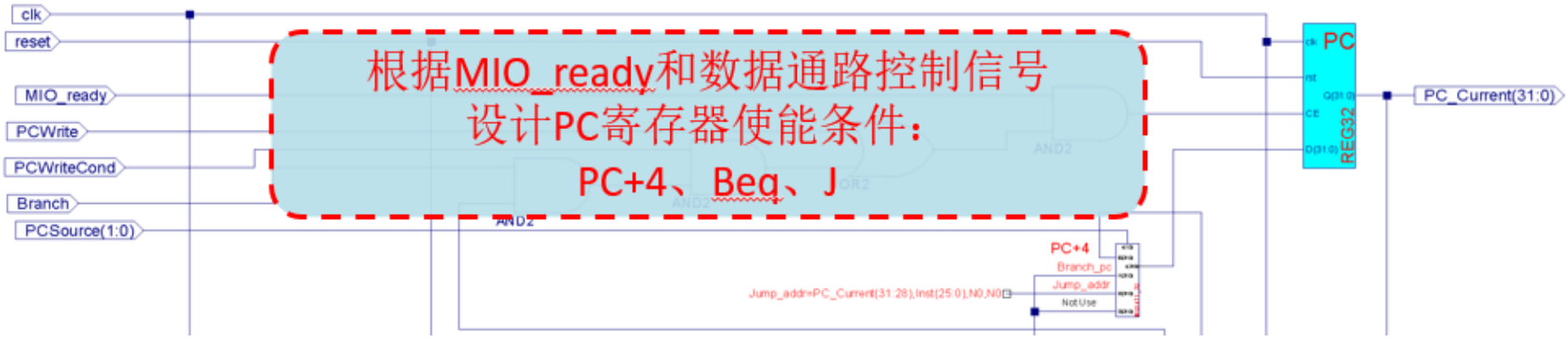
-调用实验四设计的ALU和Regs

设计工程：OExp10-MDP

- **设计CPU之数据通路**
  - 根据理论课分析讨论设计9+条指令的数据通路
  - 仿真测试M_Datapath.v模块
- **集成替换验证通过的数据通路模块**
  - 替换实验四(Exp09)中的M_Datapath.ngc核
  - 顶层模块延用Exp09
    - 模块名：Top_OExp09_MDP.v
- **测试数据通路模块**
  - 设计测试程序(MIPS汇编)测试：
  - ALU功能
  - I-指令通路
  - R-指令通路

设计要点

- 建立DataPath HDL输入模板
- 设计PC通路
  - 调用REG32和MUX4T1_32模块



  - 参考描述

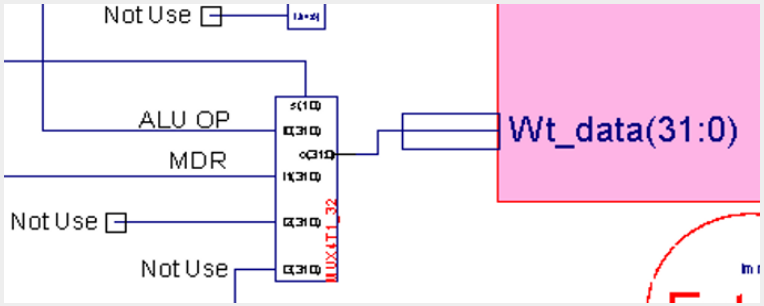
```
assign          CE = MIO_ready ?? (PCWrite || (PCWriteCond && zero&&Branch));

MUX4T1_32  MUX6(.I0(PC+4?), .I1(Beq?), .I2(Jump?), .I3(No Use), .o(PC_next));
REG32          PC(.clk(???), .rst(???), .CE(CE), .D(PC_next), .Q(PC_Current));
                ……
```

变量传输与调用Exp04的Regs模块

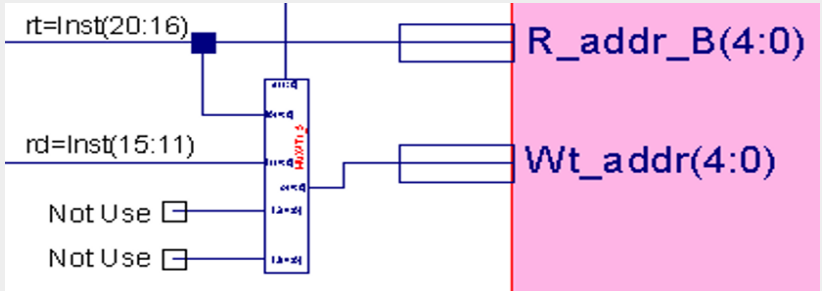
```
wire[4:0] reg_Rs_addr_A = Inst_R[25:21];   //REG Source 1  rs
wire[4:0] reg_Rt_addr_B = Inst_R[20:16];   //REG Source 2 or Destination rt
wire[4:0] reg_rd_addr = Inst_R[15:11];     //REG Destination rd
wire[15:0] imm = Inst_R[15:0];             //Immediate
wire[25:0] direct_addr = Inst_R[25:0];     //Jump addre

Regs   regs(.clk(clk),
            .rst(rst),
            .R_addr_A(reg_Rs_addr_A),
            .R_addr_B(reg_Rt_addr_B),
            .Wt_addr(??????????),      //来自MUX1输出
            .Wt_data(??????????),      //来自MUX2输出
            .L_S(RegWrite),            //来自控制器
            .rdata_A(rdata_A),         //送MUX4
            .rdata_B(rdata_B)          //送MUX3
            );
assign data_out=rdata_B;
```

寄存器Regs传输通路

```
// reg write data port
        MUX4T1_32       MUX2(.I0(ALU_Out),          //ALU OP
                             .I1(MDR),              //LW
                             .I2({32'h00000000}),   // not use
                             .I3(32'h00000000), // not use
                             .s(MemtoReg),
                              .o(w_reg_data)
                              );
```

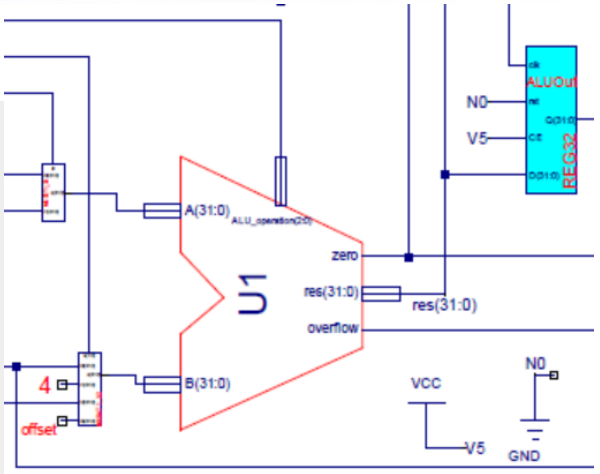



```
// reg write addr port
MUX4T1_5    MUX1(.I0(reg_Rt_addr_B),    //reg addr=IR[21:16]
                 .I1(reg_rd_addr),      //reg addr=IR[15:11], LW or lui
                 .I2(5'b11111),         // not use
                 .I3(5'b00000),         // not use
                 .s(RegDst),
                 .o(reg_Wt_addr)
                 );
```



调用Exp04的ALU模块

ALU通路及ALU模块调用描述




```
alu   x_ALU(.A(Alu_A),
            .B(Alu_B),
            .ALU_operation(ALU_operation),
            .res(res),
            .zero(zero),
            .overflow(overflow)
           );
MUX2T1_32   MUX4 (.I0(rdata_A),          // reg out A
                  .I1(PC_Current),       // PC
                  .s(ALUSrcA),
                  .o(Alu_A)
                  );
MUX4T1_32    MUX3(.I0(rdata_B),          //reg out B
                  .I1(PC+4?),            //4 for PC+4
                  .I2(???????),          //可扩展imm
                  .I3(???????),          //可扩展offset
                  .s(ALUSrcB),
                  .o(Alu_B)
                  );
REG32        ALUOut(.clk(???), .rst(???), .CE(???), .D(res), .Q(ALU_Out) );
```

指令寄存器和存储器缓冲器通路

REG32 IR(.clk(???), .rst(???), .CE(???), .D(???), .Q(**Inst**));

REG32 MDR(.clk(???), .rst(???), .CE(???), .D(???), .Q(**MDR**) **);**

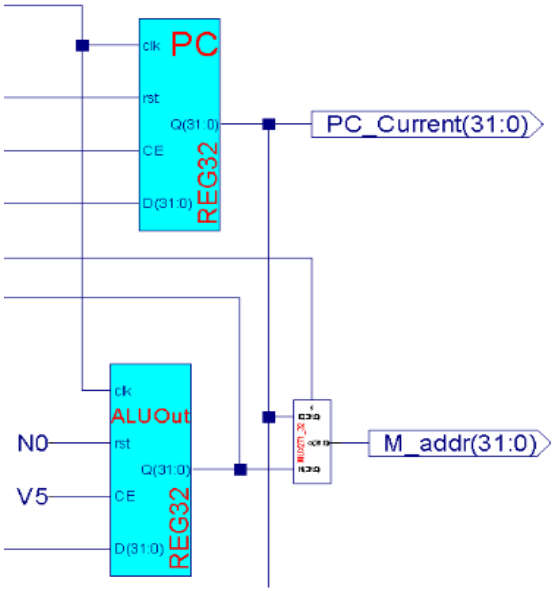
MUX2T1_32 MUX5 (.I0(ALU_Out), //access memory

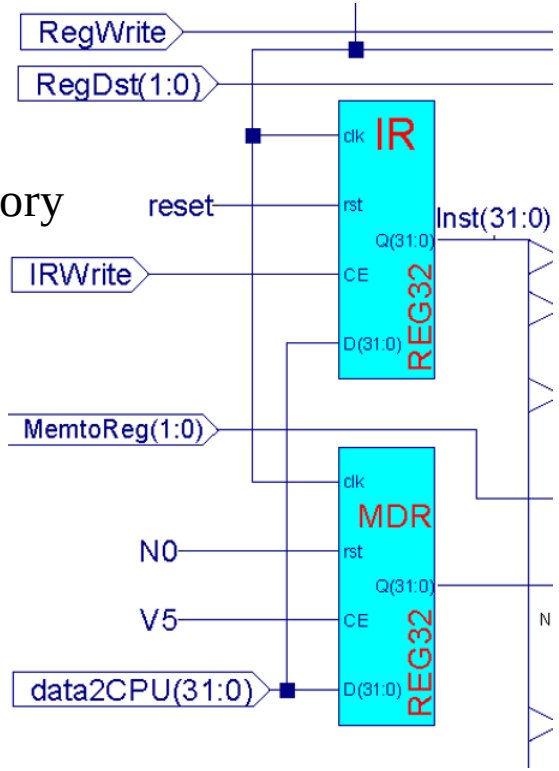
.I1(PC_Current), //IF

.s(IorD),

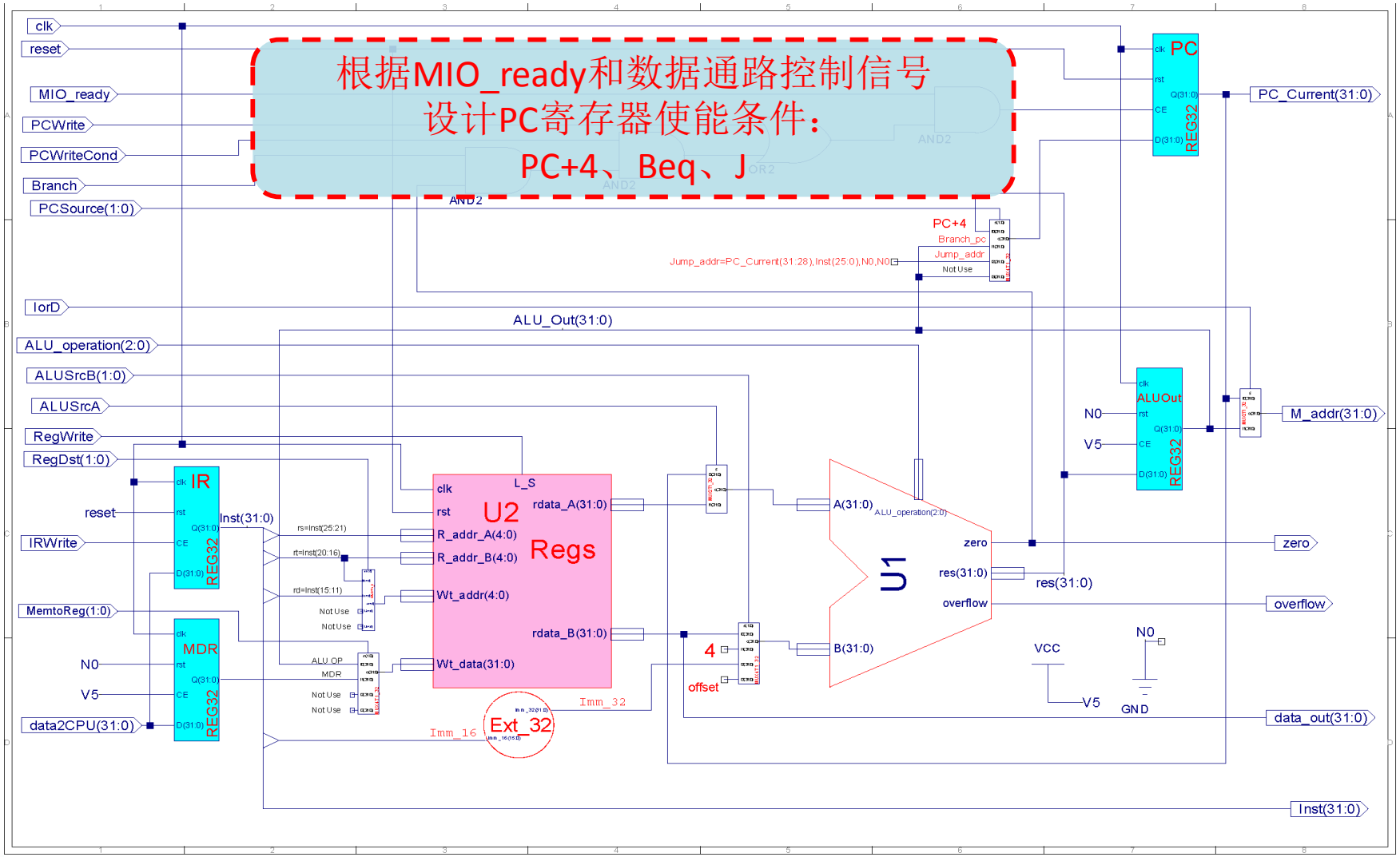
.o(M_addr)

);





完整支持9+条指令的数据通路参考描述



数据通路仿真调试

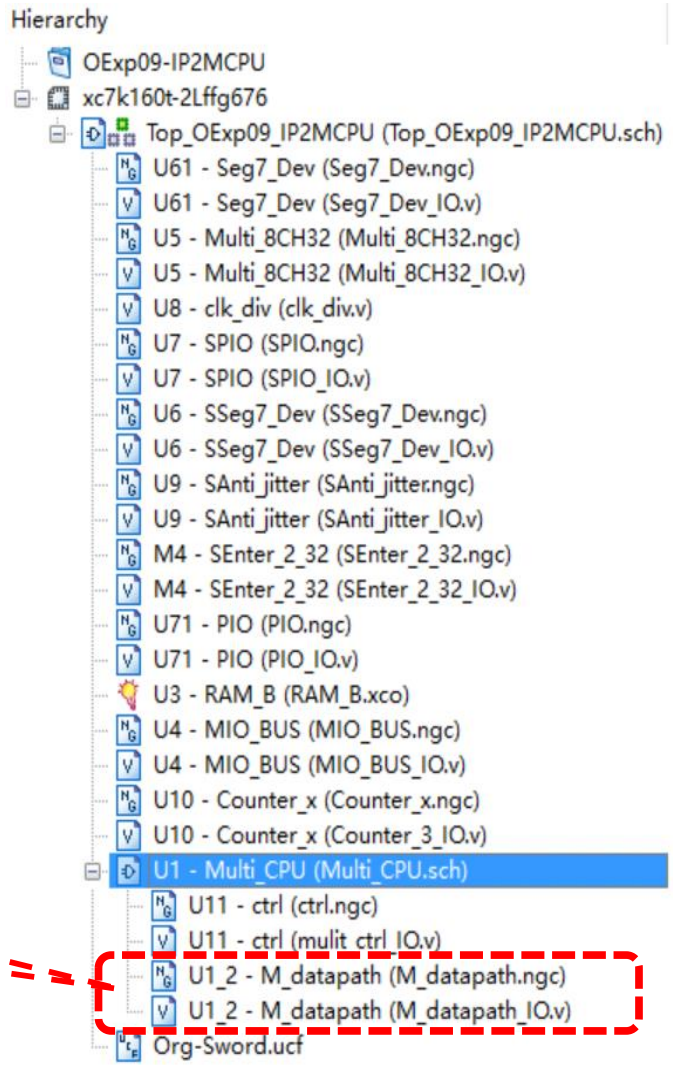
- 参考实验五，注意多周期时序
- M_Datapath模块仿真
  - 语法检查没有Errors和warnings后仿真测试
  - 仿真激励代码设计要点
    - 只做功能性测试，不做性能和完备性测试
    - **通路功能测试**
      - » 选择9条指令所有可能通路的代表指令
      - » 激励输入：
        - 计算出不同指令控制信号、代表数据和时序
        - clk、rst
    - **ALU功能测试**
      - » 选择add、and、sub、or、nor、slt指令
        - 计算出对应指令的输入控制信号和代表数据
      - » 选择Beq比较、Load和Stroe测试地址计算
    - **Regs功能测试**
      - » add指令代表作寄存器遍历测试

M_Datapath替换集成

- 集成替换
  - 仿真正确后替换Exp09的数据通路IP核
- 清理Exp09工程
  - 移除工程中的数据通路核
    - Exp09工程中移除数据通路核关联
  - 删除工程中的数据通路核文件
    - M_Datapath.ngc并替换 M_Datapath.v 文件
    - 在Project菜单中运行：

      Cleanup Project Files …
  - 建议用Exp09资源重建工程
    - 除M_Datapath.ngc核

Exp09需要清理的核



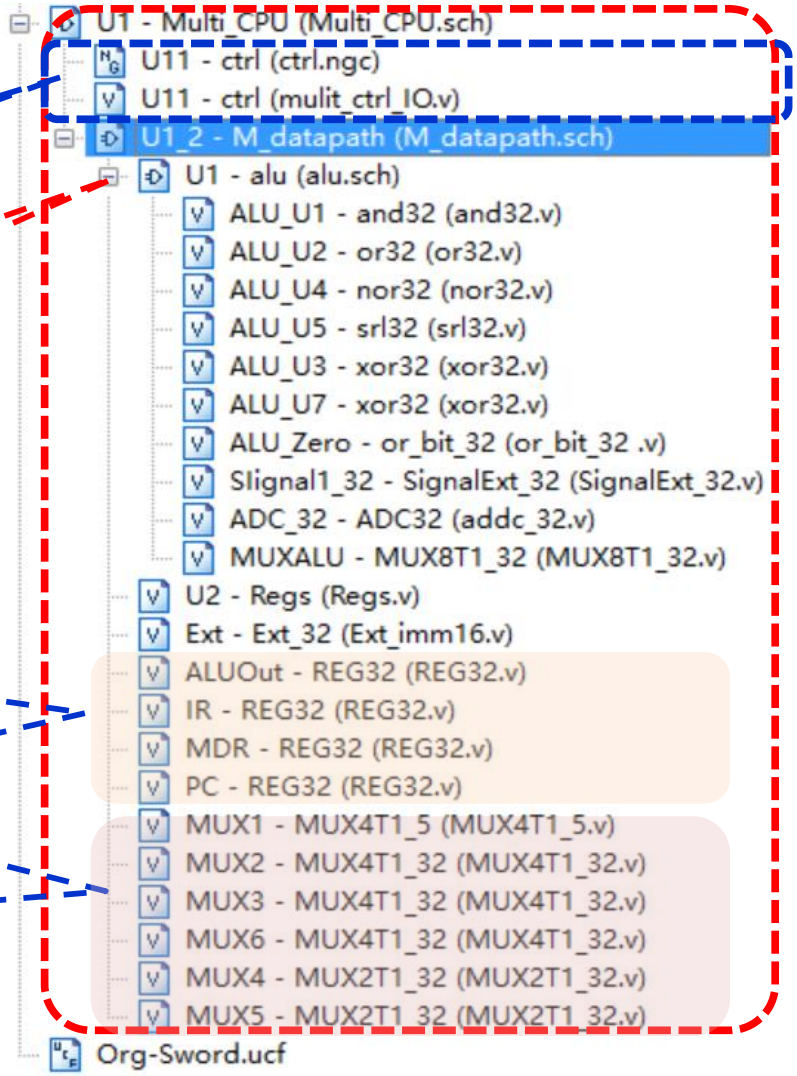
集成替换DapaPath核后的模块层次结构

控制器不变
仍然使用IP核

Exp10完成数据通路替换后的模块调用关系

需要寄存器锁在：
指令、PC计数器、存储器地址和ALU输出

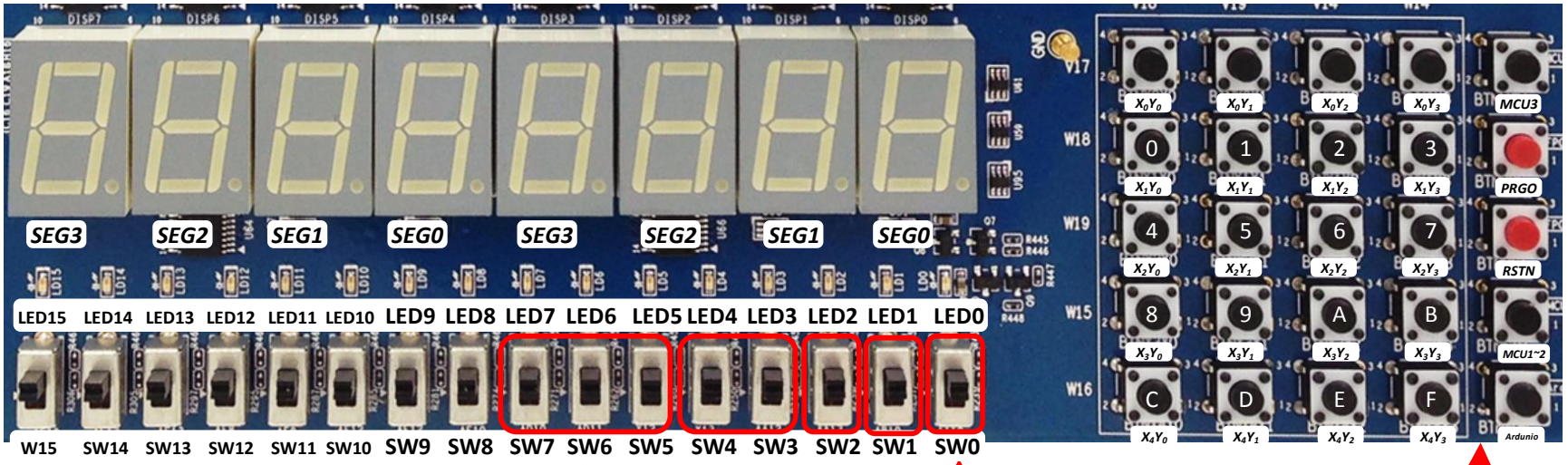
六个多路选择器：
MUX1,2：Regs输入；MUX6：PC计数器；
MUX5：存储器地址和MUX3,4：ALU输入



物理验证

- 使用DEMO程序目测数据通路功能
  - DEMO接口功能
    - SW[7:5]=000， SW[2]=0(全速运行)
      - SW[4:3]=00， SW[0]=0，点阵显示程序：跑马灯
      - SW[4:3]=00， SW[0]=0，点阵显示程序：矩形变幻
      - SW[4:3]=01， SW[0]=1，内存数据显示程序：0～F
      - SW[4:3]=10， SW[0]=1，当前寄存器R9+1显示
- 用汇编语言设计测试程序
  - 测试ALU功能
  - 测试Regs访问
  - 测试I-格式指令通路
  - 测试R-格式指令通路

物理验证-DEMO接口功能



SW[7:5]=显示通道选择
SW[7:5]=000：CPU程序运行输出
SW[7:5]=001：测试PC字地址
SW[7:5]=010：测试指令字
SW[7:5]=011：测试计数器
SW[7:5]=100：测试RAM地址
SW[7:5]=101：测试CPU数据输出
SW[7:5]=110：测试CPU数据输入

SW[0]=文本图形选择

SW[1]=高低16位选择

没有使用

SW[2]=CPU单步时钟选择

DEMO功能，测试程序可以替换成自己的功能
SW[4:3]=00，点阵显示程序：跑马灯
SW[4:3]=00，点阵显示程序：矩形变幻
SW[4:3]=01，内存数据显示程序：0～F
SW[4:3]=10，当前寄存器+1显示

测试程序参考：ALU和Regs

- 设计ALU和Regs测试程序替换DEMO程序
  - ALU、Regs测试参考设计，测试结果通过CPU输出信号单步观察
  - SW[7:5]=100，Addr_out=ALU输出
  - SW[7:5]=101，Data_out=寄存器B输出

```
#baseAddr 0000
loop:   nor r1,r0,r0;       //r1=FFFFFFFF
        slt r2,r0,r1;       //r2=00000001
        add r3,r2,r2;       //r3=00000002
        add r4,r3,r2;       //r4=00000003
        add r5,r4,r3;       //r5=00000005
        add r6,r5,r4;       //r6=00000008
        add r7,r6,r5;       //r7=0000000d
        add r8,r7,r6;       //r8=00000015
        add r9,r8,r7;       //r9=00000022
        add r10,r9,r8;      //r10=00000037
        add r11,r10,r9;     //r11=00000059
        add r12,r11,r10;    //r12=00000090
        add r13,r12,r11;    //r13=000000E9
        add r14,r13,r12;    //r14=00000179
        add r15,r14,r13;    //r15=00000262
        add r16,r15,r14;    //r16=000003DB
        add r17,r16,r15;    //r17=000006D3
        add r18,r17,r16;    //r18=00000A18
        add r19,r18,r17;    //r19=000010EB
        add r20,r19,r18;    //r20=00001B03
        add r21,r20,r19;    //r21=00003bEE
        add r22,r21,r20;    //r22=000046F1
        add r23,r22,r21;    //r23=000080DF
        add r24,r23,r22;    //r24=0000C9D0
        add r25,r24,r23;    //r25=00014AAF
        add r26,r25,r24;    //r26=0001947F
        add r27,r26,r25;    //r27=0012DF2E
        add r28,r27,r26;    //r28=001473AD
        add r29,r28,r27;    //r29=002752DB
        add r30,r29,r28;    //r30=003BC688
        add r31,r30,r29;    //r31=00621963
j loop;
```

测试程序参考

- 设计通道测试程序替换DEMO程序
  - 通道测试参考设计。测试结果通过CPU输出信号单步观察
  - 通道功能由传输数据结果来指示，如立即数通道观察：14+$zero

```
#baseAddr 0000
start:                                  //通道结果由后一条指令读操作数观察
        lw   r5, 14($zero);             //取测试常数55555555。存储器读通道
start_A:
        add r1, r5, $zero;              //r1: 寄存器写通道。R5:寄存器读通道A输出
        nor r2, $zero, r1;              //r1: 寄存器读通道B输出。R2:ALU输出通道
        lw   r5, 48($zero);  //取测试常数AAAAAAAA。立即数通道:00000048
        beq r2, r5 start_A;             //循环测试
        j     start;                    //循环测试。立即数通道：00000014
```

- 测试的完备性
  - 上述测试正确仅表明通道切换功能和总线传输部分正确
  - 要测试其完全正确，必须遍历所有可能的情况

存储器模块测试

- 设计存储器模块测试程序
  - 7段码显示器的地址是E0000000/FFFFFFE0
  - LED显示地址是F0000000/FFFFFF00
  - 请设计存储器模块测试程序
    - 测试结果在7段显示器上指示
- **RAM初始化数据：参考单周期DEMO设计**

```
memory_initialization_radix=16;
memory_initialization_vector=
00A52820, AC650000, 8C650000, 00A85824, 01A26820, 11A00017, 8C650000, 01CE9020, 0252B020,
02569020, 00B25824, 11600005, 1172000A, 01CE9020, 1172000B, AC890000, 08000036, 11410001,
0800004D, 00005027, 014A5020, AC8A0000, 08000036, 8E290860, AC890000, 08000036, 8E290820,
AC890000, 08000036, 8C0D0014, 014A5020, 01425025, 022E8820, 02348824, 01224820, 11210001,
0800005F, 000E4820, 01224820, 8C650000, 00A55820, 016B5820, AC6B0000, AC660004, 0800003E,
……
```

代码区

```
F0000000, 000002AB, 80000000, 0000003F, 00000001, FFF70000, 0000FFFF, 80000000, 00000000,
11111111, 22222222, 33333333, 44444444, 55555555, 66666666, 77777777, 88888888, 99999999,
aaaaaaaa, bbbbbbbb, cccccccc, dddddddd, eeeeeeee, FFFFFFFF, 557EF7E0, D7BDFBD9, D7DBFDB9,
DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9, FFFFBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF,
D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF, 03bdf020, 03def820, 08002300;
```

数据区

设计测试记录表格

- 学会实验数据的统计
  - 参考大学物理实验
    - 本实验没有有效数精确计算，但有大量数据表格
- ALU和Regs测试结果记录
  - 自行设计记录表格
- 通道测试结果记录
  - 自行设计记录表格
- 数据存储模块测试记录
  - 自行设计记录表格

思考题

- ALU输出为什么要锁存？
- ALU计算有效地址会与ALU正常操作冲突吗？
- 仅增加I-Type算术运算指令是否需要修改本设计的数据通路？
- 扩展下列指令，数据通路将作如何修改：

R-Type：　srl*, jr, jalr, eret*；

I-Type：　addi, andi, ori, xori, lui, bne, slti

J-Type：　Jal；


END