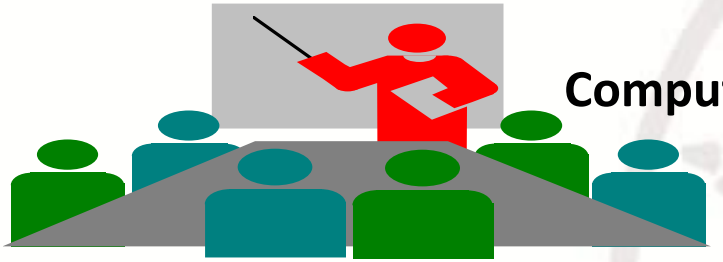


Computer Organization & Design
实验与课程设计

实验十二

多周期CPU指令扩展设计

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. 复杂时序电路实现
2. 学习CPU优化思想
3. 个性化设计探索
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

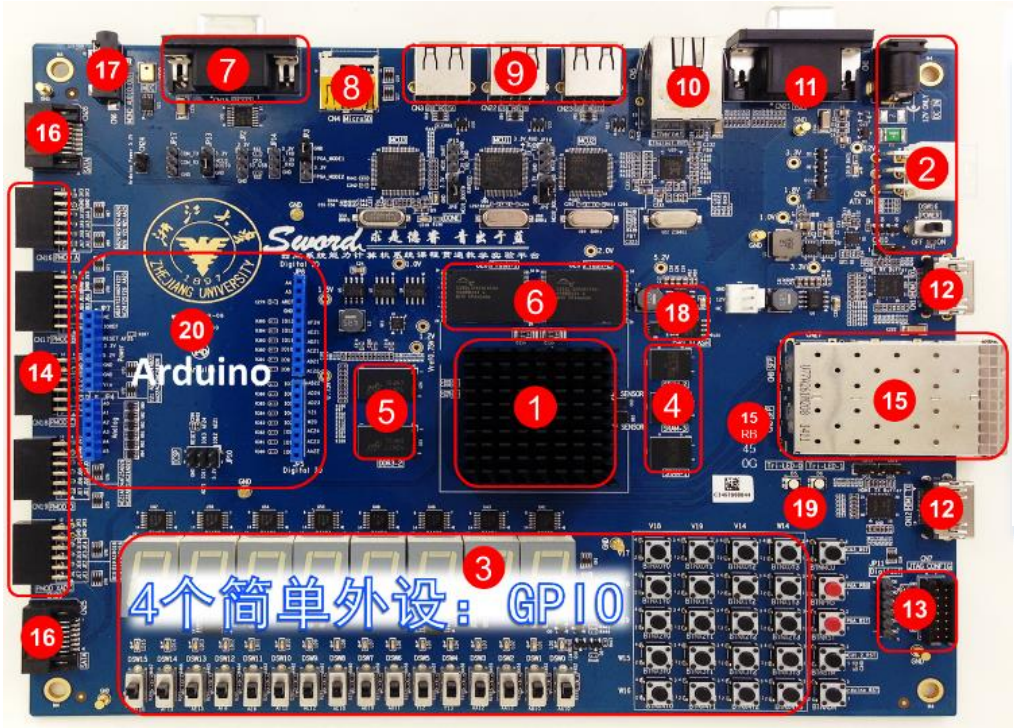
2. 计算机软硬件课程贯通教学实验系统(Sword)

3. Xilinx ISE14.4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XC7K160/325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

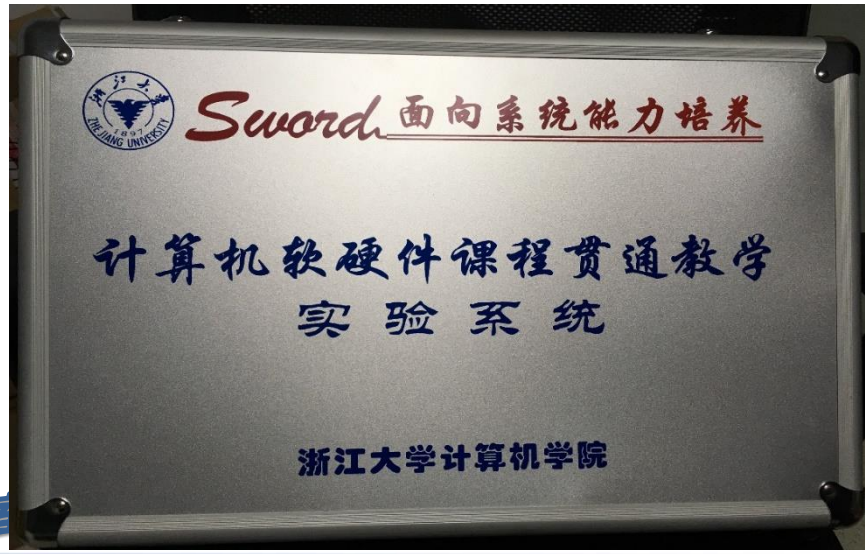
12位VGA接口（RGB656）、USB-HID（键盘）

▼ **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. 指令集优化：扩展多周期CPU指令集

- 不少于下列指令

  R-Type： add, sub, and, or, xor, nor, slt, srl*, jr, jalr, eret*；

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

  J-Type： J, Jal；

- 鼓励个性化设计
  - 必需兼容基本指令集

2. 设计CPU功能测试方案和测试例程

- 测试方案与测试程序
- 演示程序：简单有意义的DEMO
  - Project的简化版

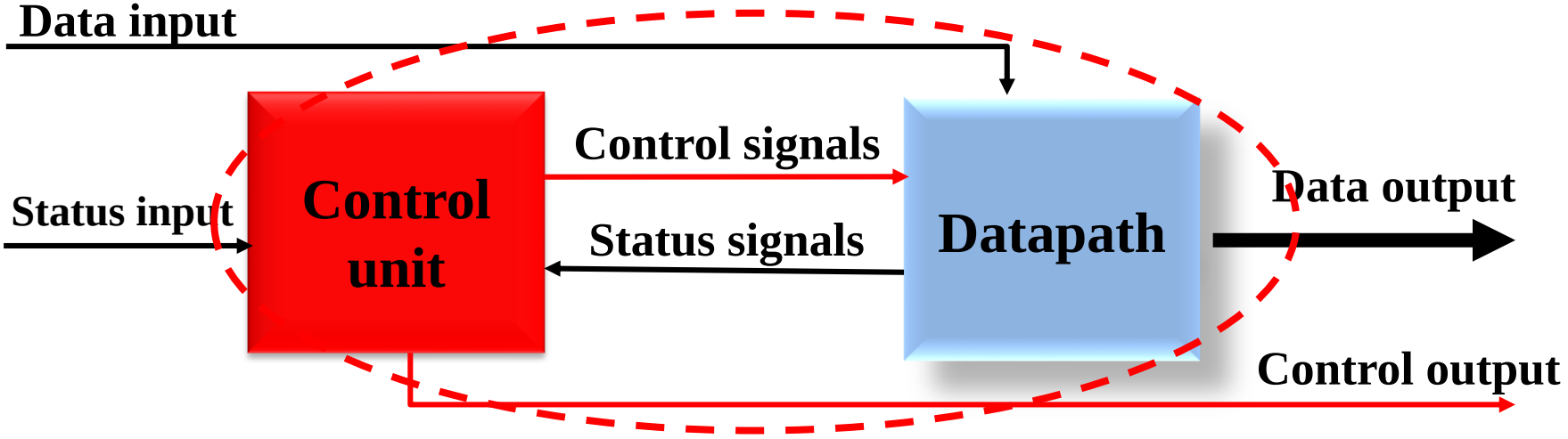
Course Outline

- 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

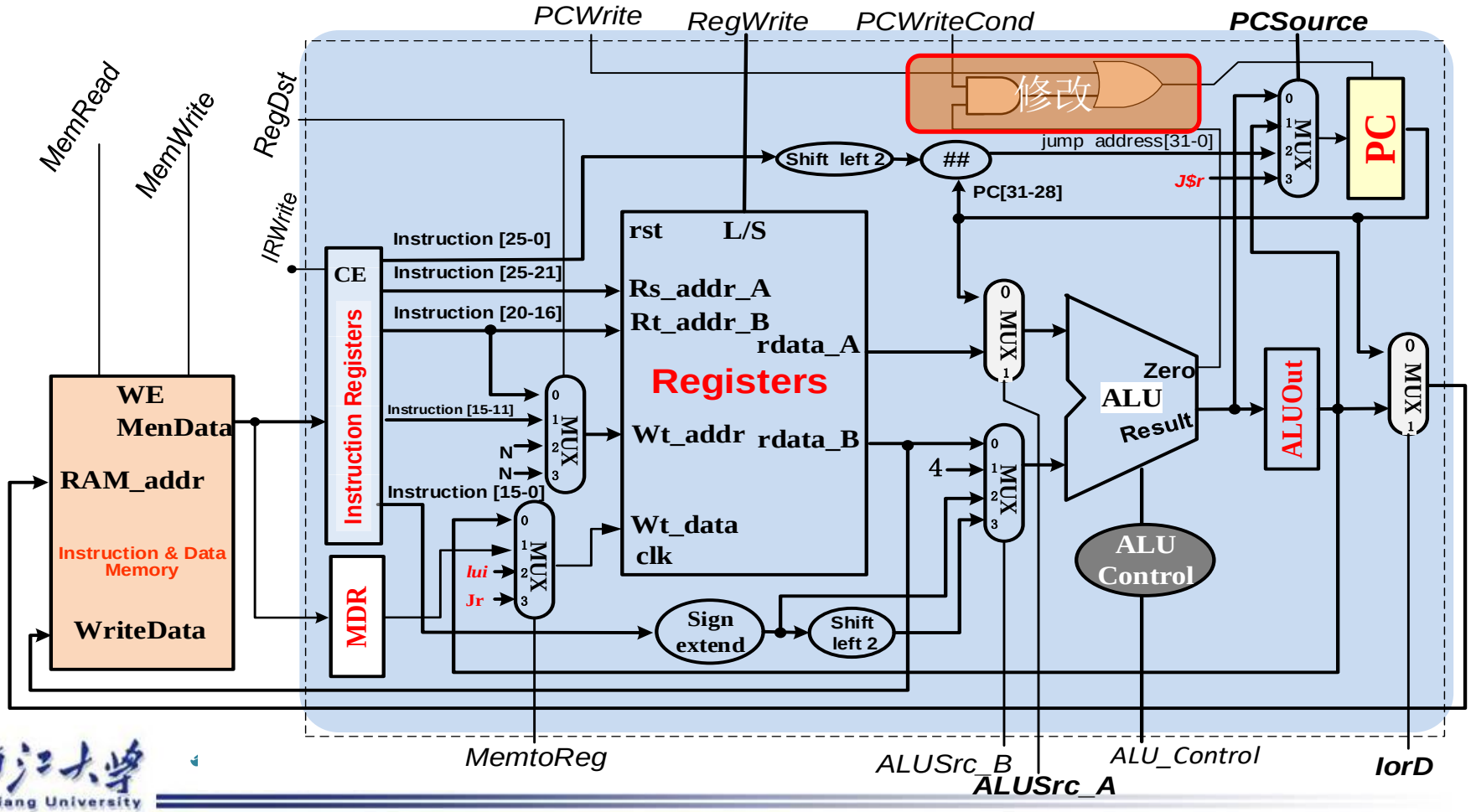


Computer organization

- Special circuits that processes logical action with instructions -Software

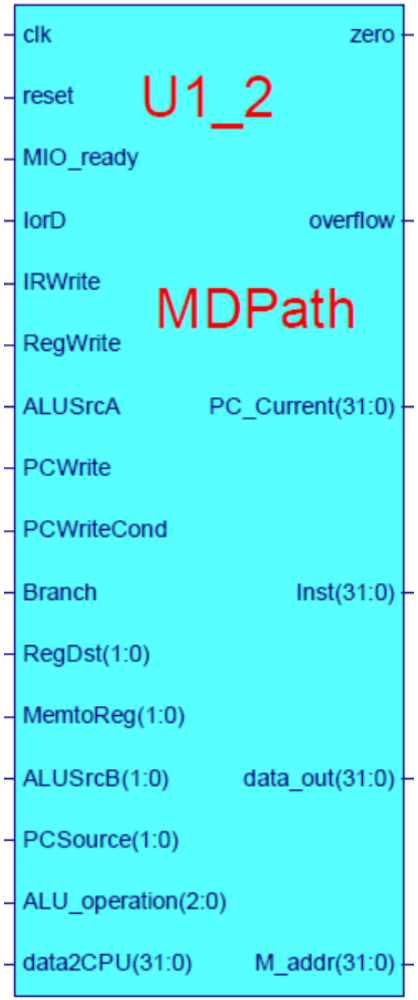
多周期数据通路结构：兼容9-23+指令

- 找出指令的通路：5+1个MUX
  - 增加或修改了什么通道？



实验12的数据通路模块：MDPath

- 重要信号
  - 已经预留了Branch
    - Branch：=1→beq；=0 → bne
    - 条件指令指示：PCWriteCond
  - 预留通道控制信号
    - **MemtoReg(1:0)**：四选一实验十只用了2路
    - **RegDst(1:0)**：四选一实验十只用了2路
    - **PCSource(1:0)**：四选一实验十只用了3路
  - 其他信号
    - Inst_R：指令寄存器输出
    - PC_Current：当前PC(PC+4)
    - M_addr：存储器地址



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
```

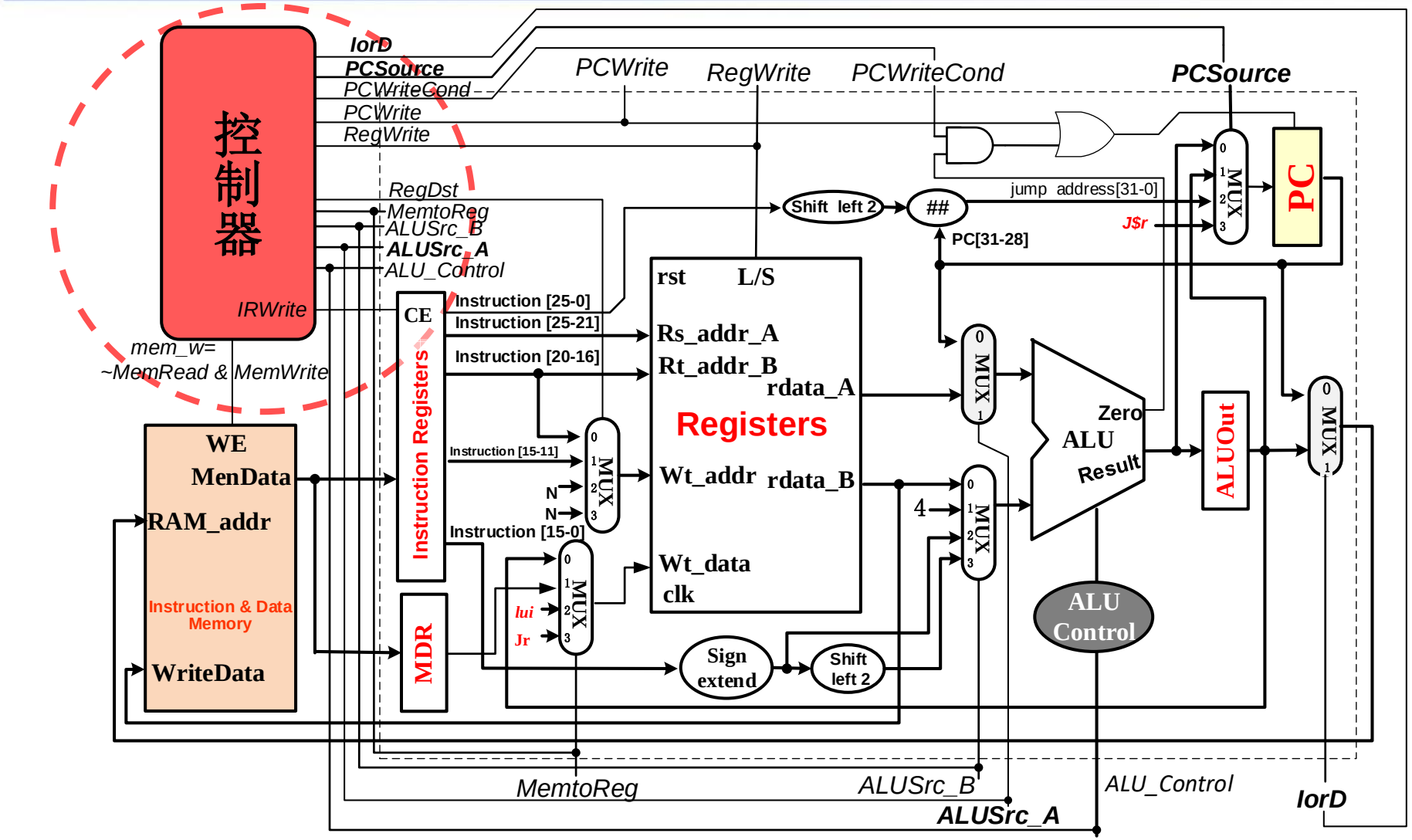
多周期控制信号定义：

- 兼容实验十基本多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	?	ALU端口A输入选择		
ALUSrc_B(1:0)	?	ALU端口B输入选择		
RegDst(1:0)	?	寄存器写地址选择		
MemtoReg(1:0)	?	寄存器写入数据选择		
IorD	?	指令或数据地址选择		
PCSource	?	Next-PC指针选择		
PCWriteCond	?	条件指令指示		
Branch	?	Beq、Bne指示		
RegWrite	-	寄存器写控制		
MemWrite	-	存储器写控制		
MemRead	-	存储器读控制		
……	?	个性化设计新增(需修改电路符号sym)		
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

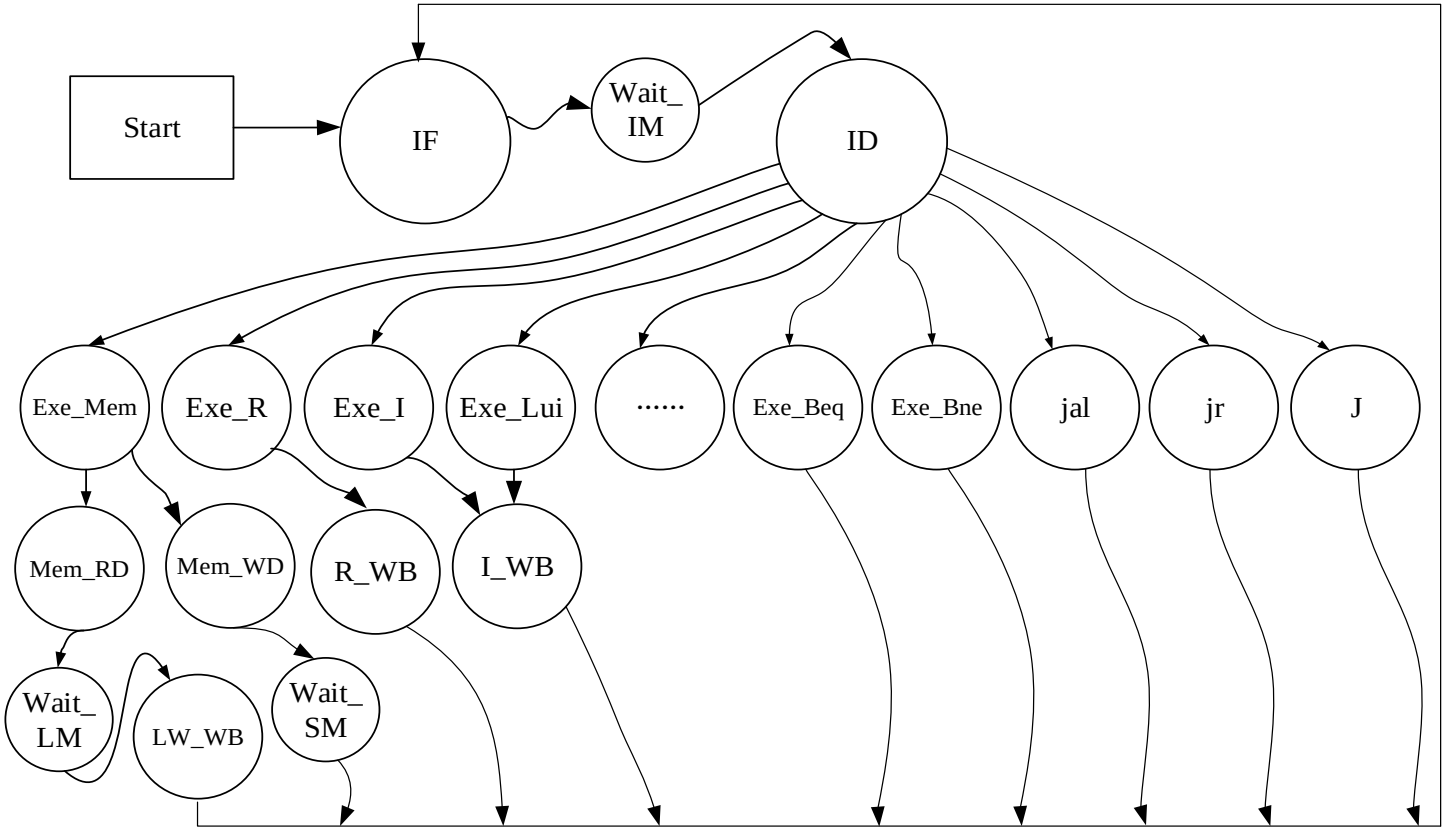
控制器与控制对象


控制器
IorD
PCSource
PCWriteCond
PCWrite
RegWrite
IRWrite
mem_w=
~MemRead & MemWrite
RegDst
MemtoReg
ALUSrc_B
ALUSrc_A
ALU_Control
PCWrite
RegWrite
PCWriteCond
PCSource
jump address[31-0]
J$r
PC
Shift left 2
##
PC[31-28]
CE
Instruction [25-0]
Instruction [25-21]
Instruction [20-16]
Instruction [15-11]
Instruction [15-0]
Instruction Registers
rst
L/S
Rs_addr_A
Rt_addr_B
rdata_A
Registers
Wt_addr
rdata_B
Wt_data
clk
N
N
4
Zero
ALU
Result
ALUOut
WE
MenData
RAM_addr
Instruction & Data Memory
WriteData
MDR
lui
Jr
Sign extend
Shift left 2
ALU Control
MemtoReg
ALUSrc_B
ALUSrc_A
ALU_Control
IorD

18+指令的状态机：根据设计指令画出

asking the students to complete the corresponding state truth table

- 根据数据通路设计所有指令状态机



分析指令控制状态机填入下表:参考实验11

状态 / 输出信号	0000 IF	0001 ID	0010 MEN-Ex	0011 MEN-RD	0100 LW_WB	0101 MEM_W	0110 R_Exc	0111 R_WB	1000 Beq_Exc	1001 J	增加	.....	.....
PCWrite													
PCWriteCond													
IorD													
MemRead													
MemWrite													
IRWrite													
MemtoReg1													
MemtoReg0													
PCSource1													
PCSource0													
ALUSrcA													
ALUSrcB1													
ALUSrcB0													
RegWrite													
RegDst1													
RegDst0													
Branch													
ALU_operation													
MEM_IO													
.....													

请分析完成输出信号真值表

九条指令的状态真值表参考

Outputs	Input values (S[3–0])									
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
PCWrite	1	0	0	0	0	0	0	0	0	1
PCWriteCond	0	0	0	0	0	0	0	0	1	0
IorD	0	0	0	1	0	1	0	0	0	0
MemRead										0
MemWrite										0
IRWrite	1	0	0	0	0	0	0	0	0	0
MemtoReg	0	0	0	0	1	0	0	0	0	0
PCSource1	0	0	0	0	0	0	0	0	0	1
PCSource0	0	0	0	0	0	0	0	0	1	0
ALUOp1										
ALUOp0										
ALUSrcB1	0	1	1	0	0	0	0	0	0	0
ALUSrcB0	1	1	0	0	0	0	0	0	0	0
ALUSrcA	0	0	1	0	0	0	1	0	1	0
RegWrite	0	0	0	0	1	0	0	1	0	0
RegDst	0	0	0	0	0	0	0	1	0	0

建议直接输出存储器写控制信号：men_w

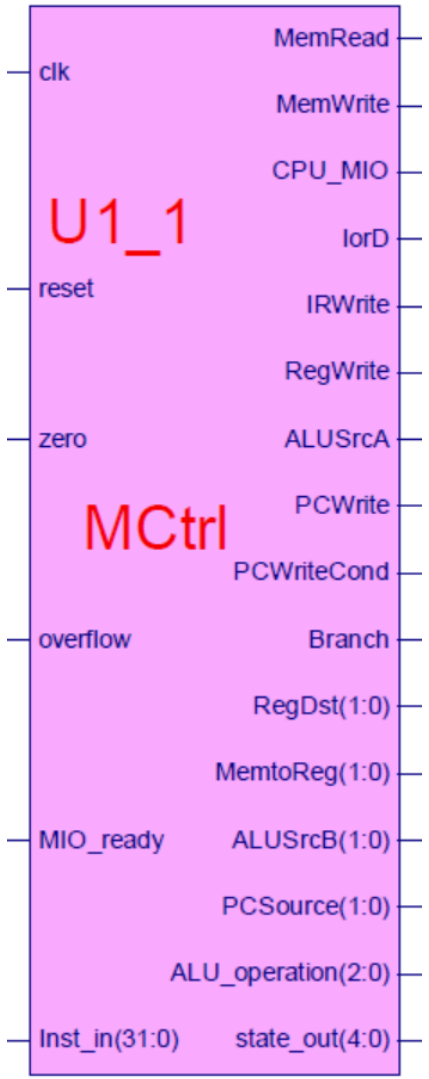
建议直接输出ALU控制信号：ALU_operation

控制器实现

- 控制器实现方案
  - 指令实现建议采用一级译码方案
    - 主控制器直接输出ALU_operation
    - ALU译码电路仍可调用单周期设计模块
- 状态机实现方法
  - 建议根据状态表HDL直接描述：
    - 与实际工程设计一样，便于优化与扩展
    - 建议输出信号与状态机分离描述
      - 结构清楚
      - HDL直接描述
  - 注意：状态机与输出信号混合描述有一个时钟的差异

多周期控制器：MCtrl

- 重要信号：兼容实验11
  - 已经预留了Branch
    - Branch：=1→beq；=0 → bne
    - 条件指令指示：PCWriteCond
  - 预留通道控制信号
    - **MemtoReg(1:0)**：四选一实验十只用了2路
    - **RegDst(1:0)**：四选一实验十只用了2路
    - **PCSource(1:0)**：四选一实验十只用了3路
- 其他信号
  - MIO_ready：外设就绪
    - =0 CPU等待；=1 CPU正常运行
    - 本实验恒等于1
  - CPU_MIO：CPU访问存储器指示
  - Inst_R：指令寄存器输出
  - PC_Current：当前PC(PC+4)
  - M_addr：存储器地址
  - State_out：状态编码，用于测试



兼容实验十的控制器接口- ctrl.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,              //实验十预留到2位
             output reg RegWrite,
             output reg [1:0]MemtoReg,            //实验十预留到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-存储器初始化数据文档：mem.coe
代码与数据共存

memory_initialization_radix=16;

memory_initialization_vector=

```
3c03f000, 2014003f, 3c088000, 00632020, 20020001, 00000827, 00205020, 20070003, 00e73827, 20067fff,
00008820, 200502ab, ac650000, 20120002, ac600004, 8c650000, 00a52820, 00a52820, ac650000, ac660004,
3c0dffff, 8c650000, 00a52820, 00a52820, ac650000, 8c650000, 00a85824, 21ad0001, 11680015, 8c650000,
20120018, 00b25824, 11600005, 1172000a, 20120008, 1172000b, ac890000, 08000015, 11410001, 0800002a,
00005027, 014a5020, ac8a0000, 08000015, 8e2902a0, ac890000, 08000015, 8e290260, ac890000, 08000015,
3c0dffff, 014a5020, 01425025, 22310004, 02348824, 21290001, 11210001, 0800003b, 21290005, 8c650000,
00a55820, 016b5820, ac6b0000, ac660004, 8c650000, 00a85824, 1168fffd, 0800001d, 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;
```

数据区：地址起始需要约定

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

设计工程：OExp12-MSOC

◎扩展多周期CPU不少于下列指令

R-Type： add, sub, and, or, xor, nor, slt, srl*, jr, jalr, eret；

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

J-Type： J, Jal*；

◎集成替换验证通过的新CPU

- 替换实验11(Exp11)中的ctrl模块
- 替换实验11(Exp11)中的M_Datapath模块
- 顶层模块延用Exp11
  - 模块名：OExp11_MCPU/MSOC.sch

◎测试扩展后的CPU功能

- 设计测试程序(MIPS汇编)测试

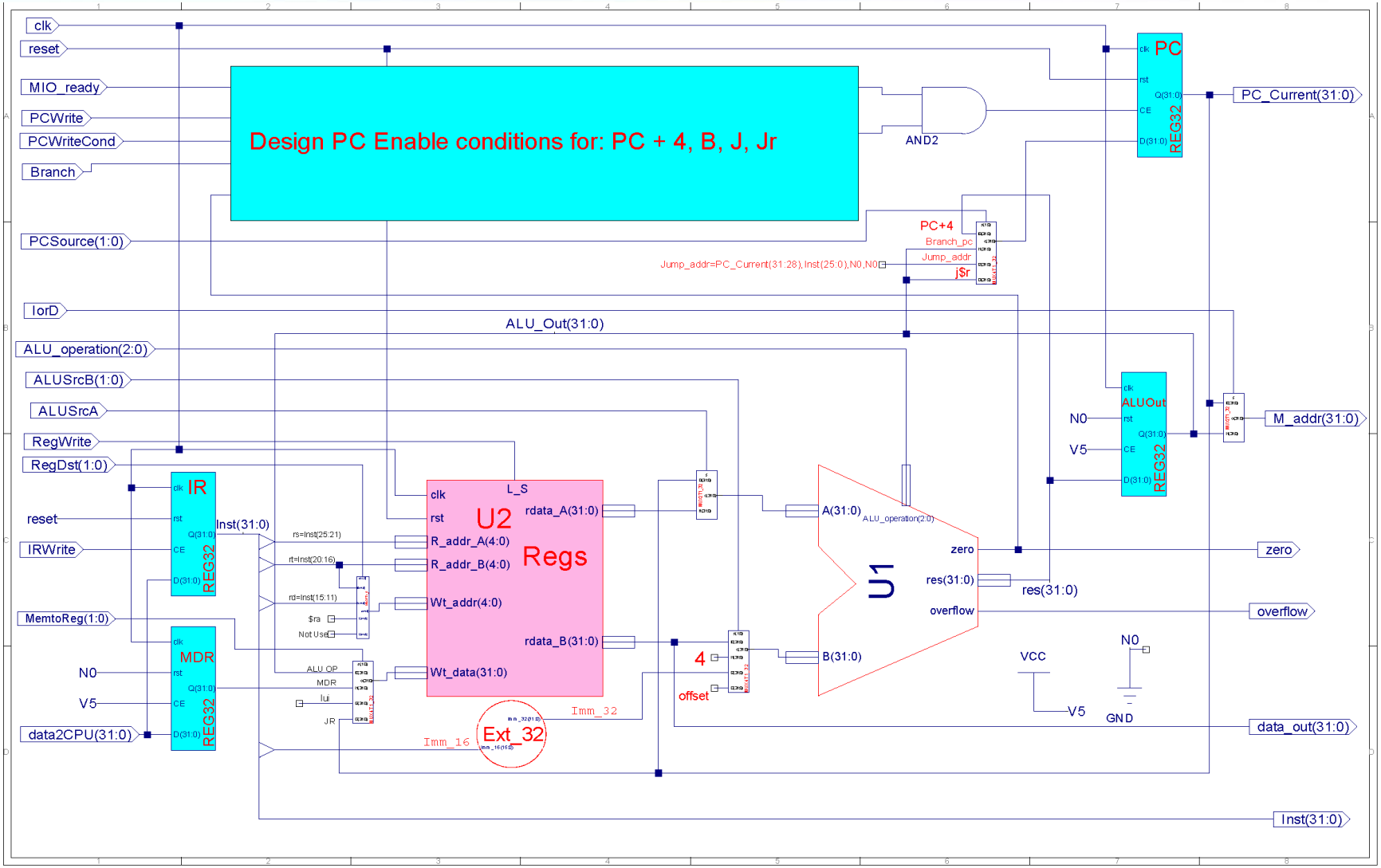
多周期CPU扩展设计

数据通路与控制器设计

设计要点

- **设计指令扩展后的多周期CPU数据通路**
  - 根据实验11的数据通路做兼容修改
  - 实现数据通路并仿真验证数据通路
- **设计指令扩展后的多周期CPU控制器**
  - 根据新数据通路及指令执行流程设计状态图
  - 根据状态图完成状态真值表和输出信号真值表
  - 根据状态表实现控制器和输出电路
    - 直接HDL结构化描述实现
  - 仿真测试控制器模块
- **集成替换验证后的控制器模块**
  - 替换实验11(Exp11)中的ctrl.v模块
  - 顶层模块延用Exp11：模块名：OExp11_MCPU.v
- **测试控制器模块**
  - 设计测试程序(MIPS汇编)测试：
  - OP译码测试：R-格式、访存指令、分支指令，转移指令

指令扩展后M_Datapath参考设计



23条指令的状态真值表-仅供参考

输出控制信号	输入 Q_3 Q_2 Q_1 Q_0（当前状态—现态）															
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	IF	ID	Mem_Ex	Mem_RD	LW_WB	Mem_WD	R_Exe	R_WB	Beq_Exe	J	I_Exe	I_WB	Lui_WB	Bne_Exe	Jr	Jal
PCWrite	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1
PCWriteCond	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
IorD	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
MemRead	1	0														0
MemWrite	0															
IRWrite	1															
MemtoReg1	0															
MemtoReg0	0															
PCSource1	0															
PCSource0	0															
ALUSrcB1	0															
ALUSrcB0	1															
ALUSrcA	0															
RegWrite	0															
RegDst1	0															
RegDst0	0															
Mem/-IO	1	0														0
ALUOp1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
ALUOp0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0
	12821	00060	00050	06001	00208	05001	00010	0001a	08090	10160	00050	00058	00468	08090	10010	1076c

首先完成状态真值表，然后设计状态机
由于信号定义和指令状态机划分不同结果也会不同

控制器HDL直接描述结构

主控制器状态机描述结构

```
parameter IF =   5'b00000,    ID=5'b00001,      EX_R=  5'b00010, EX_Mem=5'b00011, EX_I=  5'b00100,
          Lui_WB=5'b00101, EX_beq=5'b00110, EX_bne= 5'b00111, EX_jr= 5'b01000,    EX_JAL=5'b01001,
          Exe_J = 5'b01010,   MEM_RD=5'b01011,                MEM_WD= 5'b01100,
          WB_R=  5'b01101,  WB_I=5'b01110,    WB_LW=5'b01111, Error=11111;
parameter AND=3'b000, OR=3'b001, ADD=3'b010, SUB=3'b110,
          NOR=3'b100, SLT=3'b111, XOR=3'b011, SRL=3'b101;
`define CPU_ctrl_signals  {PCWrite, PCWriteCond, IorD, MemRead, MemWrite, IRWrite, MemtoReg, PCSource,
                           ALUSrcB, ALUSrcA, RegWrite, RegDst, CPU_MIO, ……}
```

```
always @ (posedge clk or posedge reset)begin
                ……
end

always @* begin
                …… 输出变量(信号)描述
                …… ALU操作控制描述
end
```

状态机

数据通路控制(含ALU)

ALU操作译码描述

状态机转换描述

```
always @ (posedge clk or posedge reset)
    if (reset==1) state <= IF;
     else
        case (state)
        IF: if(MIO_ready) state <= ID;
            else state <= IF;
        ID: case (Inst_in[31:26])
                6'b000000: state <= state <= EX_R;    //R-type OP
                6'b100011: state <= EX_Mem;           //Lw
                      ••••••••••••••••••
                6'b000100: state <= EX_beq;           //Beq
                6‘b000101: state <= EX_bne;           //Bne
                      ••••••••••••••••••
                default:      state <= Error;
                endcase
        Mem_Ex:begin
                                  ••••••••••••
        Error:     state <= Error;
        default:  state <= Error;
        endcase
```

输出变量(信号)描述： 指令扩展后value需要重新定义值

```
always @ * begin
    case(Q)                                          //state
        IF:        begin ` CPU_ctrl_signals = value0; ALU_operation = ADD;  end
        ID:        begin ` CPU_ctrl_signals = value1; ALU_operation = ADD;  end
        EX_Mem:  begin ` CPU_ctrl_signals = value2; ALU_operation = ADD; end
        EX_R:      begin ` CPU_ctrl_signals = value6;
                    case (Inst_in[5:0])
                     6‘b100000:  ALU_operation = ???? ;
                     6‘b100010:  ALU_operation = ???? ;
                                 ..........
                     default:      ALU_operation  <= ADD;
                    endcase
                    end
        MEM_RD: begin ` CPU_ctrl_signals = value3; ALU_operation = ADD; end
        WB_LW :   begin ` CPU_ctrl_signals = value4; ALU_operation = ADD; end
                                 ......................
        default:     begin ` CPU_ctrl_signals = value0; ALU_operation = ADD; end
    endcase
```

R-Type

状态EX_R时ALU操作译码描述参考

- R格式指令ALU译码参考描述

```
EX_R:  begin ` CPU_ctrl_signals = value6;
    case (Inst_in[31:26])                          //R-type OP
        6'b100000: ALU_operation  <= ？ ;
        6'b100010: ALU_operation  <= ?;
        6'b100100: ALU_operation  <= ?;
        6'b100101: ALU_operation  <= ?;
        6'b100111: ALU_operation  <= ?;
        6'b101010: ALU_operation  <= ?;
        6'b000010: ALU_operation  <= ?;        //SP3 shfit 1bit right
        6'b000000: ALU_operation  <= ?;
        6'b001000: ALU_operation  <=ADD;   //Jr
        default:     ALU_operation  <= ADD;
        endcase
      end
```

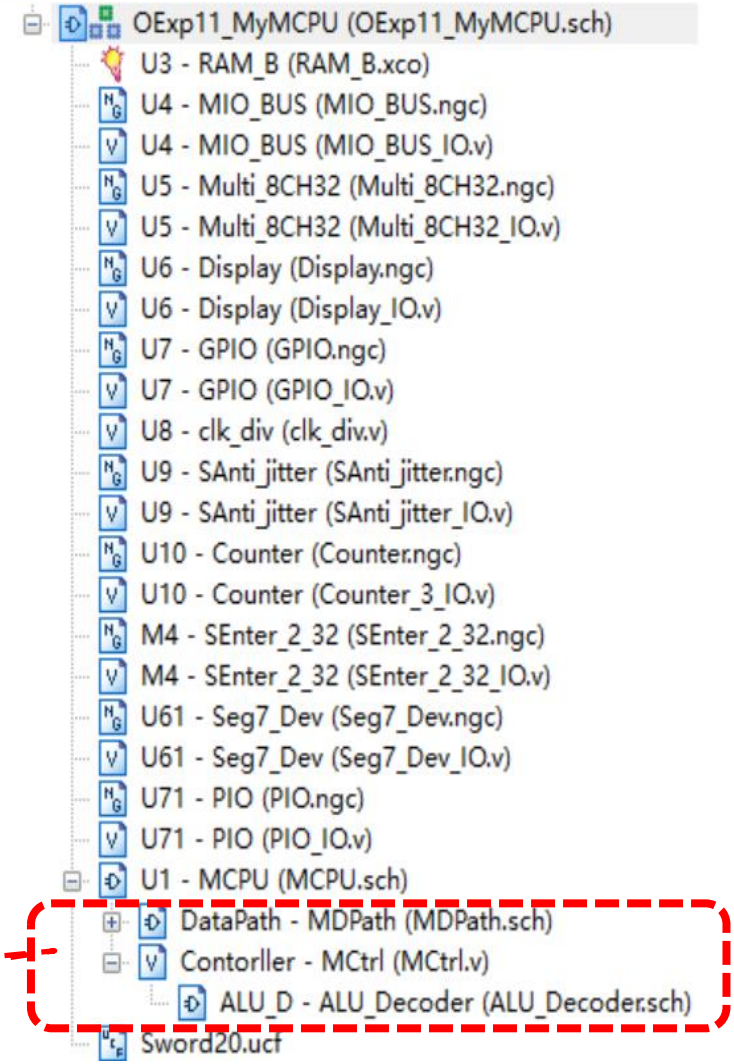
- 输出信号与状态时序
  - 输出信号若与状态机合并描述有一个时钟差

多周期CPU扩展实现

集成替换

控制器集成替换

- 集成替换
  - 仿真正确后替换Exp11工程中的
    - 数据通路模块、控制器模块
- **移除工程中数据通路与控制器关联**
  - Exp11工程中移除数据通路关联
  - Exp11工程中移除控制器关联
  - 替换工程中数据通路模块文件
    - M_Datapath.v
  - 替换工程中控制器模块文件
    - ctrl.v文件
  - 在Project菜单中运行：
    Cleanup Project Files …
  - 建议用Exp11资源重建工程
    - 除MDpath.v和MCtrl.v模块

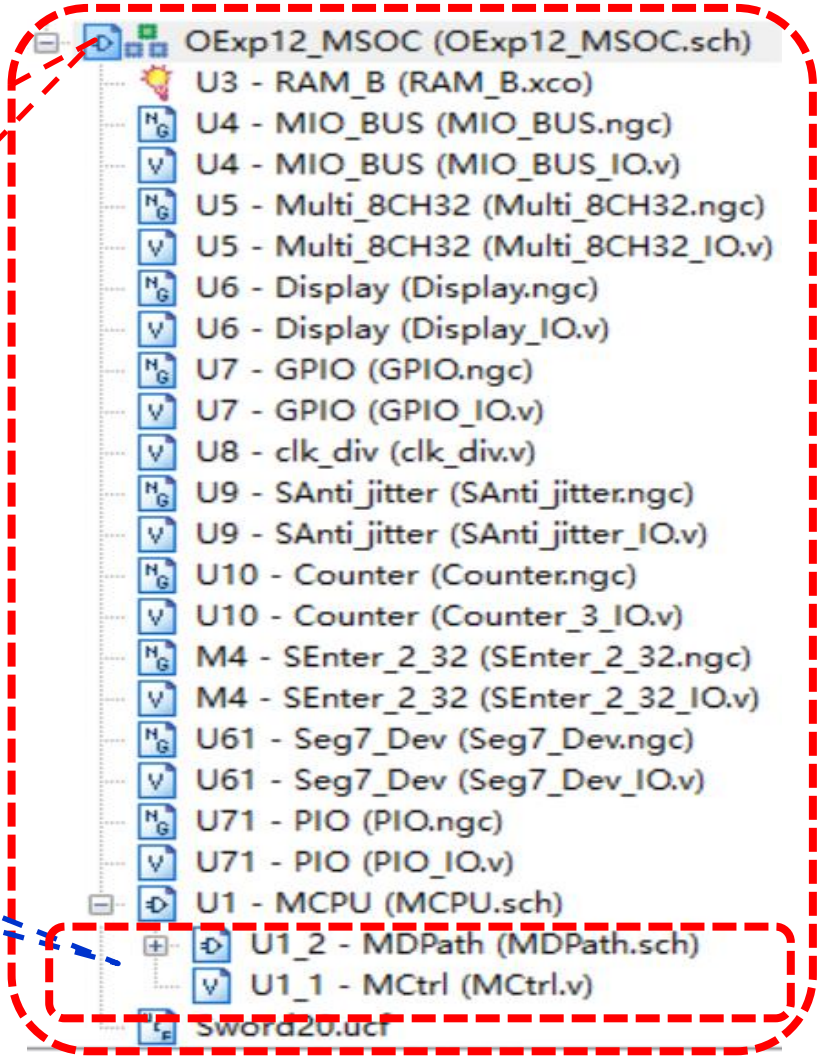


Exp11需要替换的模块

集成替换后的模块层次结构

Exp12完成数据通路替换后的模块调用关系

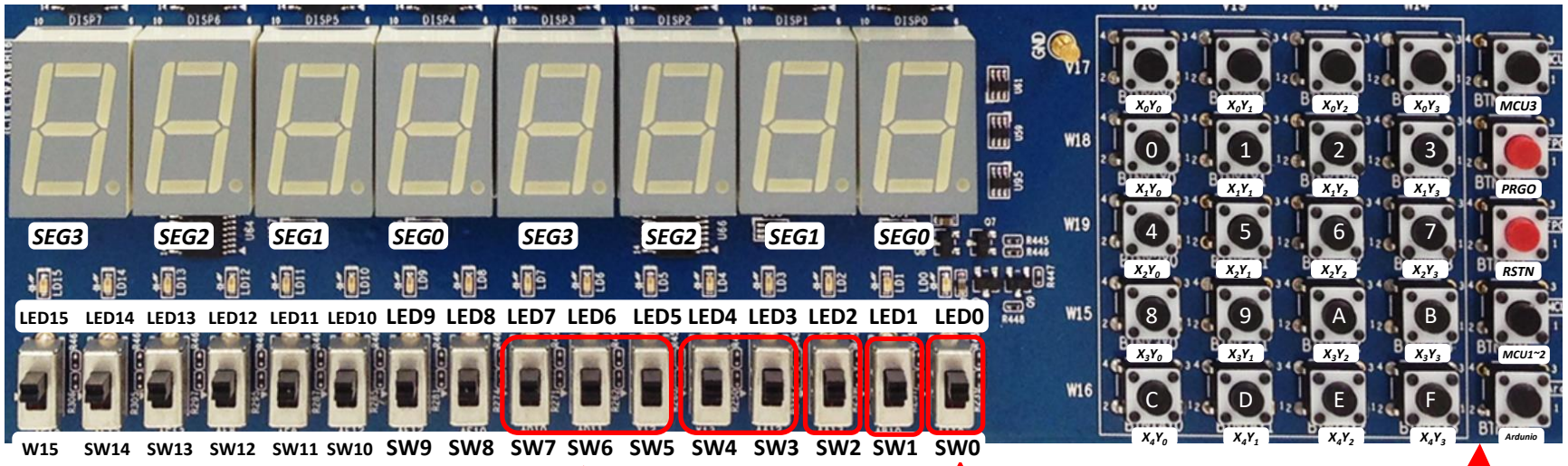
替换后的数据通路与控制器模块



物理验证

- 使用DEMO程序目测控制器功能
  - DEMO接口功能： (同实验11，代码不同)
    - 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指令(R-格式译码、 Function译码)
  - 测试LW/SW指令(I-格式译码)
  - 测试分支指令(I-格式译码)
  - 测试转移指令(J-格式译码)

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

- **设计CPU功能测试方案和测试例程**
  - 物理调试正确后，设计测试方案与测试程序
  - 演示程序：简单有意义的DEMO
    - 修改DEMO兼有测试、自检并有一定趣味
    - Project的简化版


END