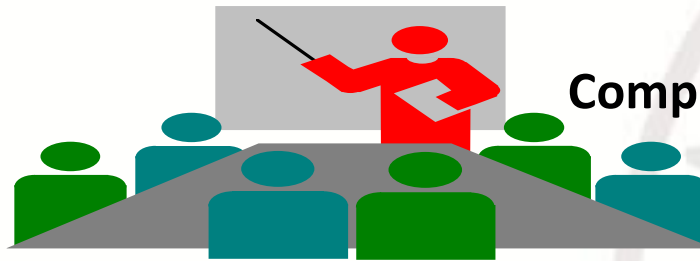




浙江大学  
ZHEJIANG UNIVERSITY



Computer Organization & Design

# Computer Organization & Design

## 实验与课程设计

### 实验十一

### 多周期CPU设计-控制器设计

施青松

Asso. Prof. Shi Qingsong

College of Computer Science and Technology, Zhejiang University

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

---

# Course Outline





# 实验目的

1. 深入运用寄存器传输控制技术
2. 掌握CPU核心：指令执行过程与控制流关系
3. 设计多周期数据通路控制器
4. 测试方案的设计
5. 测试程序的设计



# 实验环境

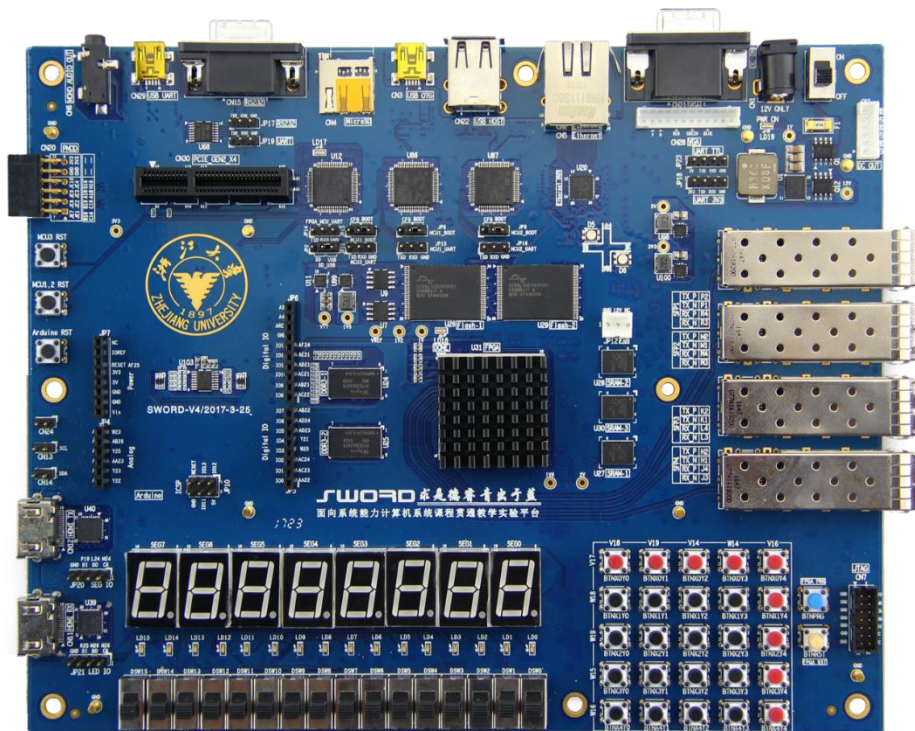
## □ 实验设备

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

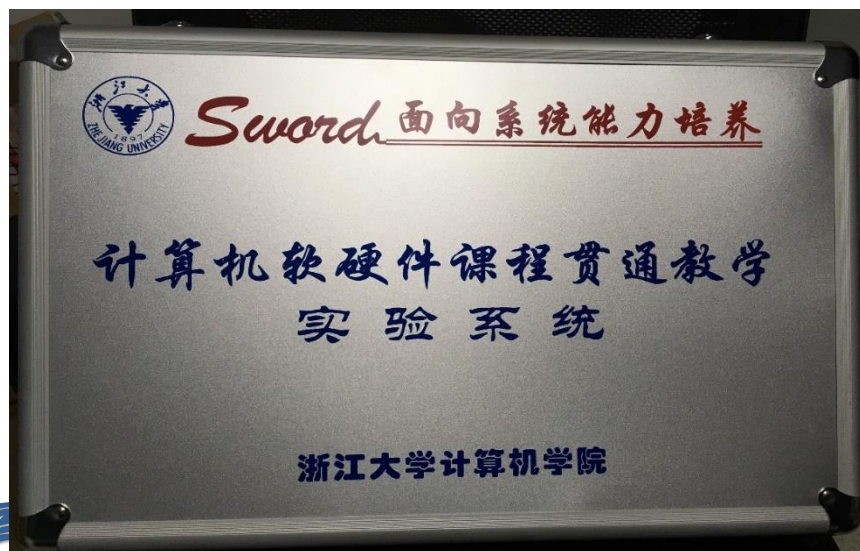
12位VGA接口 (RGB656)、USB-HID (键盘)

### ▼ 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





## 1. 设计9+条指令的控制器

- 用硬件描述语言设计实现控制器
  - 此实验在Exp10的基础上完成

## 2. 设计控制器测试方案：

- OP译码测试：R-格式、访存指令、分支指令，转移指令
- 运算控制测试：Function译码测试

## 3. 设计控制器测试程序

# Course Outline

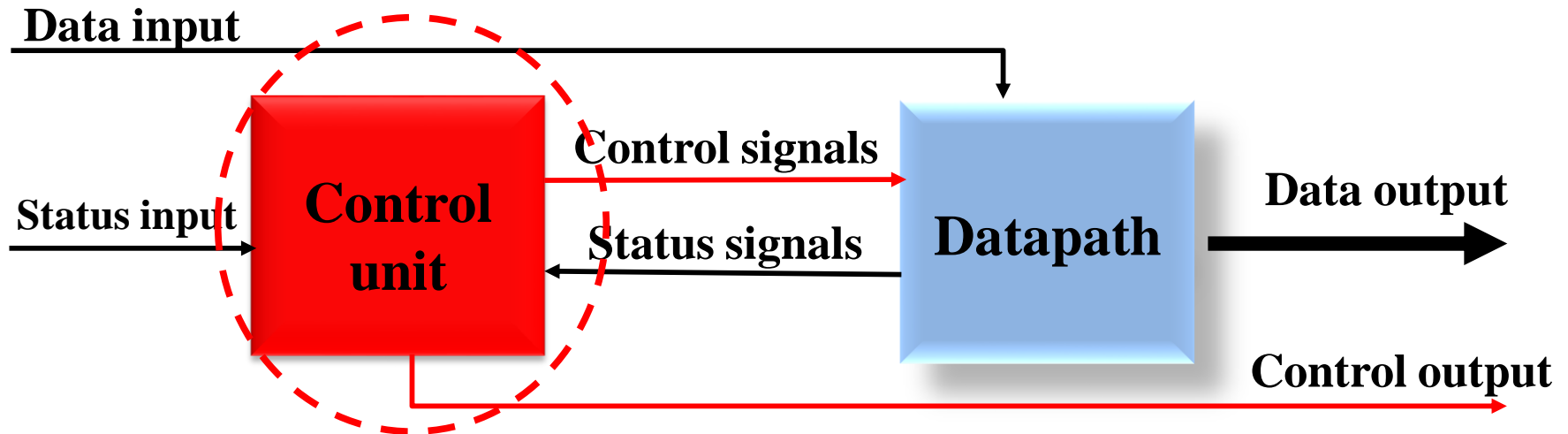




# CPU organization

## □ Digital circuit

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



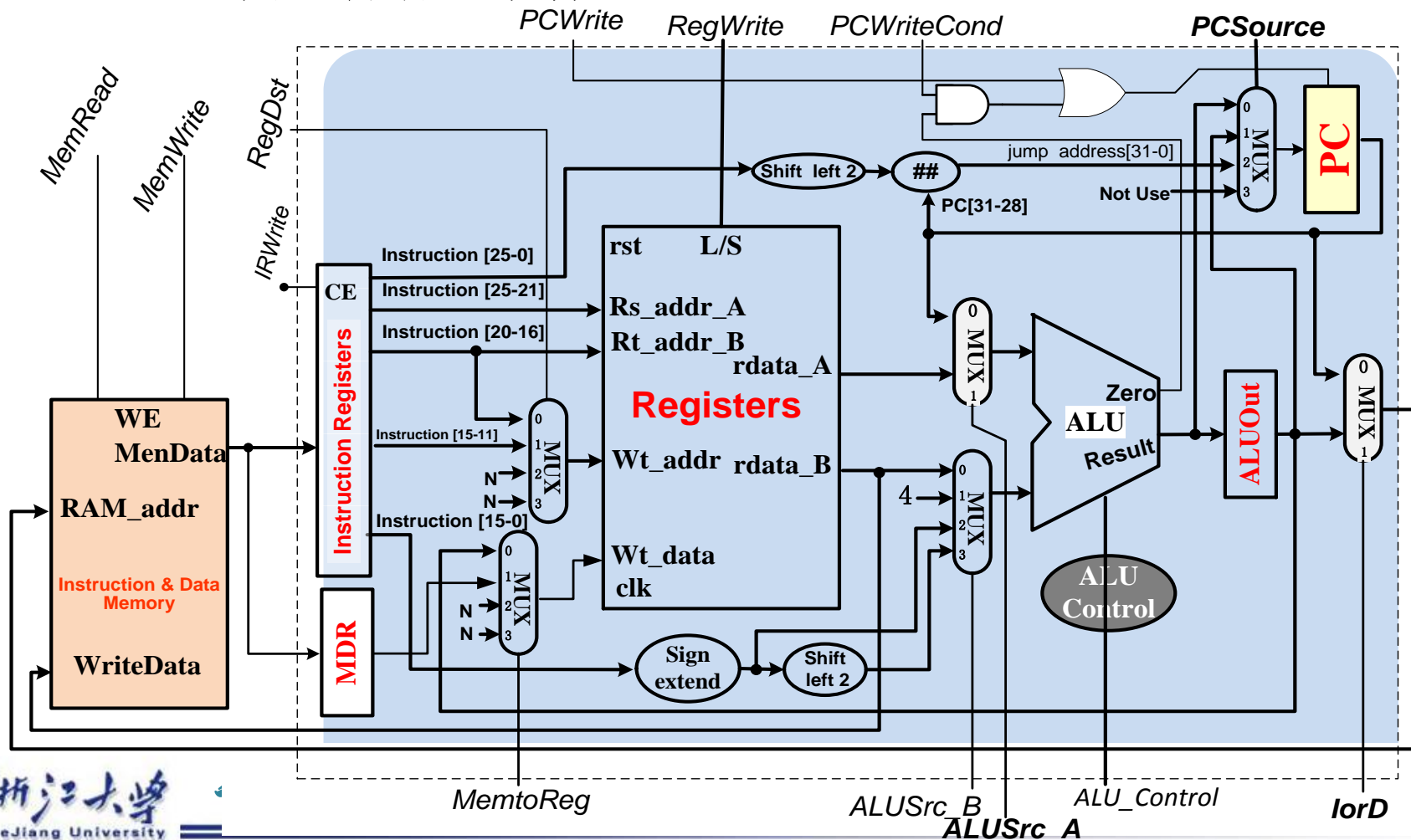
## □ Computer organization

- 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

■ 比单周期增加了什么通道？



# 多周期数据通路模块：MDPPath



## □ 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

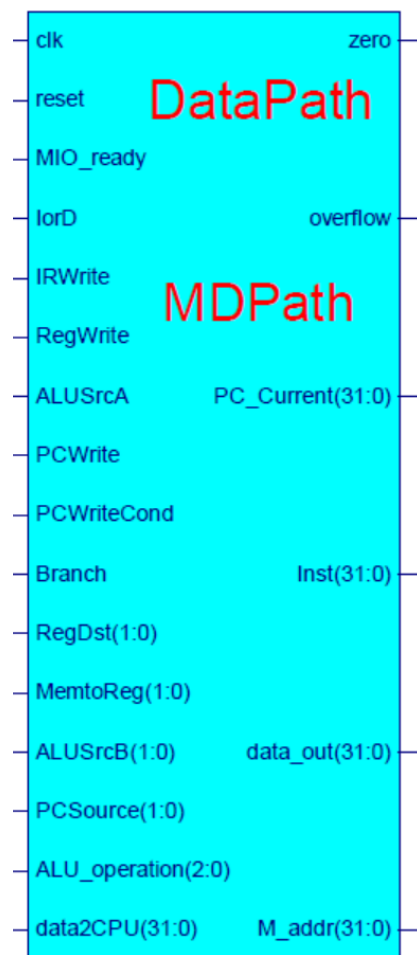


# 数据通路接口参考- MDPath.v

```
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] PCSrc,          //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
```





# 控制器设计方案

## □ 控制器实现有多种方法

### ■ 状态转换实现:

- 状态表→状态方程→激励方程→HDL描述
- 或状态表→HDL行为描述

### ■ 激励方程和输出信号:

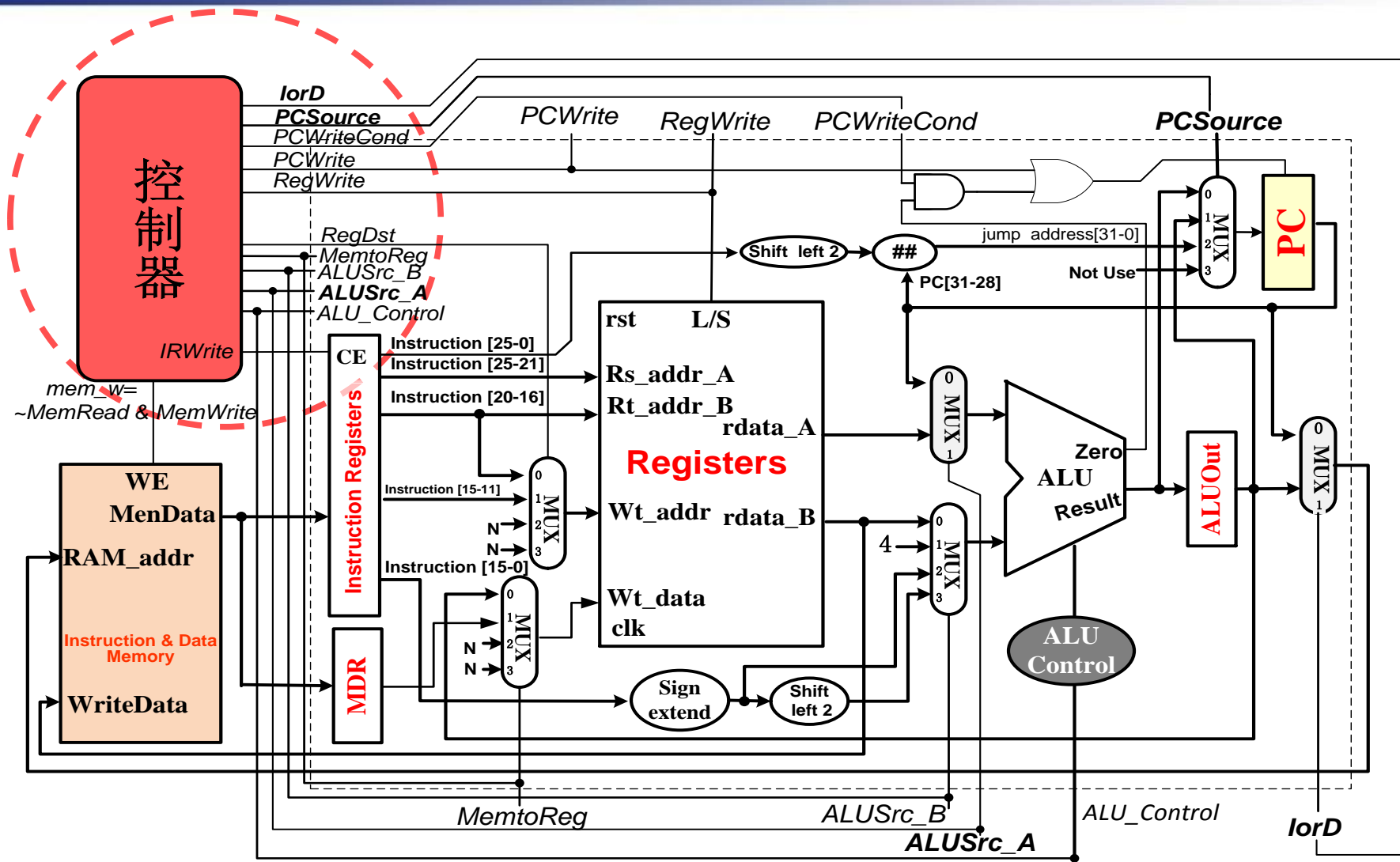
- HDL直接描述
- ROM/ PLA (教材光盘)
- MUX
- 门电路

## □ 这里根据**时序电路的一般设计流程分析**

### ■ 实现时可以选用任意一种方法。建议:

- 9+条指令不是非常复杂, 用激励方程HDL描述实现
- 指令较多时用HDL直接描述状态表实现

# 控制器与控制对象

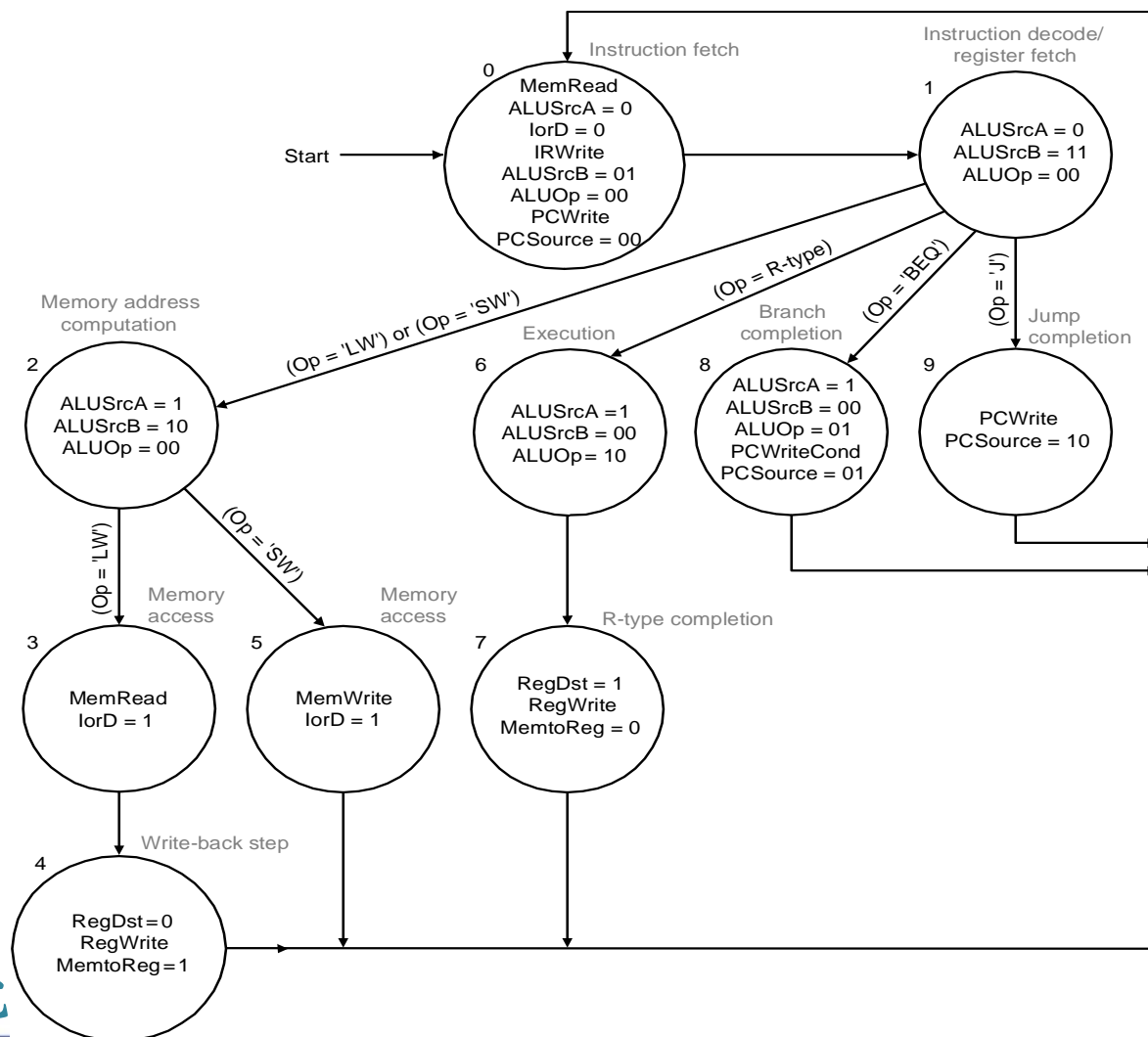


# 9+指令的状态机：根据设计指令画出

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





# 状态编码与分配

- 根据状态图可知10个状态需要4位状态变量表示
- 实验选用4个D触发器存储状态变量
- 状态编码有多种编码方法，不唯一。这里采用与教材一致的顺序状态编码，不是最佳方案：

状态	触发器状态分配	备注
	$Q_{3n}$ $Q_{2n}$ $Q_{1n}$ $Q_{0n}$	
0	0 0 0 0	
1	0 0 0 1	
2	0 0 1 0	
3	0 0 1 1	
4	0 1 0 0	
5	0 1 0 1	
6	0 1 1 0	
7	0 1 1 1	
8	1 0 0 0	
9	1 0 0 1	



# 状态转换表/次态表

- 根据状态图和输入变量 $OP_0 \sim OP_5$ 写出状态转换表
  - 4个状态变量共有16个状态，其中1010~1111六个状态为非工作状态
  - 操作码有6个变量，共 $2^6=64$ 个最小项组合，只有5种组合为有效输入，其余为无效输入，可作任意项考虑。

序号	现 态	输 入(指令操作码)	次 态	备注
	$Q_{3n} Q_{2n} Q_{1n} Q_{0n}$	$Op_5 Op_4 Op_3 Op_2 Op_1 Op_0$	$Q_{3n+1} Q_{2n+1} Q_{1n+1} Q_{0n+1}$	
0	0 0 0 0	x x x x x x	0 0 0 1	1 Op无关
1	0 0 0 1	0 0 0 0 0 0	0 1 1 0	6 R-type
		1 0 x 0 1 1	0 0 1 0	2 L/S
		0 0 0 1 0 0	1 0 0 0	8 Beq
		0 0 0 0 1 0	1 0 0 1	9 Jump
2	0 0 1 0	1 0 0 0 1 1	0 0 1 1	3 Load
		1 0 1 0 1 1	0 1 0 1	5 Store
3	0 0 1 1	1 0 0 0 1 1	0 1 0 0	4 Load
4	0 1 0 0	1 0 0 0 1 1	0 0 0 0	0 Load
5	0 1 0 1	1 0 1 0 1 1	0 0 0 0	0 Store
6	0 1 1 0	0 0 0 0 0 0	0 1 1 1	7 R-type
7	0 1 1 1	0 0 0 0 0 0	0 0 0 0	0 R-type
8	1 0 0 0	0 0 0 1 0 0	0 0 0 0	0 Beq
9	1 0 0 1	0 0 0 0 1 0	0 0 0 0	0 Jump



# 状态方程

□ 根据状态转换表可以写出状态方程如下：

$$Q_{3n+1} = \text{State1} (\text{Beq} + \text{Jump})$$

$$Q_{2n+1} = \text{State1} \text{ Rtype} + \text{State2} \text{ Store} + \text{State3} \text{ Load} + \text{State6} \text{ Rtype}$$

$$Q_{1n+1} = \text{State1} (\text{Rtype} + \text{LS}) + \text{State2} \text{ Load} + \text{State6} \text{ Rtype}$$

$$Q_{0n+1} = \text{State0} + \text{State1} \text{ Jump} + \text{State2} \text{ Load} + \text{State2} \text{ Store} + \text{State6} \text{ Rtype}$$

$$Q_{3n+1} = \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}Q_{0n}(\bar{O}p_5\bar{O}p_4\bar{O}p_3O p_2\bar{O}p_1\bar{O}p_0 + \bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2O p_1\bar{O}p_0)$$

$$Q_{2n+1} = \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}Q_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0 + \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}\bar{Q}_{0n}O p_5\bar{O}p_4O p_3\bar{O}p_2O p_1O p_0 + \\ \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}Q_{0n}O p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2O p_1O p_0 + \bar{Q}_{3n}Q_{2n}Q_{1n}\bar{Q}_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0$$

$$Q_{1n+1} = \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}Q_{0n}(\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0 + O p_5\bar{O}p_4X\bar{O}p_2O p_1O p_0) + \\ \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}\bar{Q}_{0n}O p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2O p_1O p_0 + \bar{Q}_{3n}Q_{2n}Q_{1n}\bar{Q}_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0$$

$$Q_{0n+1} = \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}\bar{Q}_{0n} + \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}Q_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2O p_1\bar{O}p_0 + \\ \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}\bar{Q}_{0n}O p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2O p_1O p_0 + \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}\bar{Q}_{0n}O p_5\bar{O}p_4O p_3\bar{O}p_2O p_1O p_0 + \\ \bar{Q}_{3n}Q_{2n}Q_{1n}\bar{Q}_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0$$

□  $O p_4$ 全是“0”可以简化，但意义不大



# 激励方程

## □ 根据D触发器特征方程和状态方程可得D触发器激励函数：

$$D_3 = \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}Q_{0n}(\bar{O}p_5\bar{O}p_4\bar{O}p_3Op_2\bar{O}p_1\bar{O}p_0 + \bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2Op_1\bar{O}p_0)$$

$$D_2 = \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}Q_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0 + \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}\bar{Q}_{0n}Op_5\bar{O}p_4Op_3\bar{O}p_2Op_1Op_0 + \\ \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}Q_{0n}Op_5\bar{O}p_4\bar{O}p_3\bar{O}p_2Op_1Op_0 + \bar{Q}_{3n}Q_{2n}Q_{1n}\bar{Q}_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0$$

$$D_1 = \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}Q_{0n}(\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0 + Op_5\bar{O}p_4X\bar{O}p_2Op_1Op_0) + \\ \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}\bar{Q}_{0n}Op_5\bar{O}p_4\bar{O}p_3\bar{O}p_2Op_1Op_0 + \bar{Q}_{3n}Q_{2n}Q_{1n}\bar{Q}_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0$$

$$D_0 = \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}\bar{Q}_{0n} + \bar{Q}_{3n}\bar{Q}_{2n}\bar{Q}_{1n}Q_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2Op_1\bar{O}p_0 + \\ \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}\bar{Q}_{0n}Op_5\bar{O}p_4\bar{O}p_3\bar{O}p_2Op_1Op_0 + \bar{Q}_{3n}\bar{Q}_{2n}Q_{1n}\bar{Q}_{0n}Op_5\bar{O}p_4Op_3\bar{O}p_2Op_1Op_0 + \\ \bar{Q}_{3n}Q_{2n}Q_{1n}\bar{Q}_{0n}\bar{O}p_5\bar{O}p_4\bar{O}p_3\bar{O}p_2\bar{O}p_1\bar{O}p_0$$

$$D_3 = State1 (Beq + Jump)$$

$$D_2 = State1 Rtype + State2 Store + State3 Load + State6 Rtype$$

$$D_1 = State1 (Rtype + LS) + State2 Load + State6 Rtype$$

$$D_0 = State0 + State1 Jump + State2 Load + State2 Store + State6 Rtype$$

## □ 这一步完成了状态转换的设计

- 根据D触发器激励方程可以画出状态转换逻辑
- 现代工程设计已经不需要自己求解

### □ EDA综合器会自动综合

# 状态激励表

- 也可以根据D触发器特征和状态表得到状态激励表
  - 和状态表类同，仅输出是触发器输入D，一般用状态方程配方
  - 输出真值信号太多，需要另列
    - 采用Moore状态机，输出仅与状态有关

序号	现 态	输 入(指令操作码)	D触发器输入	输出	备注
	$Q_{3n} Q_{2n} Q_{1n} Q_{0n}$	Op5 Op4 Op3 Op2 Op1 Op0	$D_3 D_2 D_1 D_0$	另列	
0	0 0 0 0	x x x x x x	0 0 0 1		Op无关
1	0 0 0 1	0 0 0 0 0 0	0 1 1 0		R-type
		1 0 x 0 1 1	0 0 1 0		L/S
		0 0 0 1 0 0	1 0 0 0		Beq
		0 0 0 0 1 0	1 0 0 1		Jump
2	0 0 1 0	1 0 0 0 1 1	0 0 1 1		Load
		1 0 1 0 1 1	0 1 0 1		Store
3	0 0 1 1	1 0 0 0 1 1	0 1 0 0		Load
4	0 1 0 0	1 0 0 0 1 1	0 0 0 0		Load
5	0 1 0 1	1 0 1 0 1 1	0 0 0 0		Store
6	0 1 1 0	0 0 0 0 0 0	0 1 1 1		R-type
7	0 1 1 1	0 0 0 0 0 0	0 0 0 0		R-type
8	1 0 0 0	0 0 0 1 0 0	0 0 0 0		Beq
9	1 0 0 1	0 0 0 0 1 0	0 0 0 0		Jump



# 输出信号真值表(状态激励表另列部分)

□ 根据状态图和多周期数据通路控制要求信号真值表如下:

输入 $Q_{3n} Q_{2n} Q_{1n} Q_{0n}$ (当前状态—现态)										输出控制信号
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	
1	0	0	0	0	0	0	0	0	1	PCWrite
0	0	0	0	0	0	0	0	1	0	PCWriteCond
0	0	0	1	0	1	0	0	0	0	IorD
1	0	0	1	0	0	0	0	0	0	MemRead
0	0	0	0	0	1	0	0	0	0	MemWrite
1	0	0	0	0	0	0	0	0	0	IRWrite
0	0	0	0	1	0	0	0	0	0	MemtoReg
0	0	0	0	0	0	0	0	0	1	PCSource1
0	0	0	0	0	0	0	0	1	0	PCSource0
0	0	0	0	0	0	1	0	0	0	ALUOp1
0	0	0	0	0	0	0	0	1	0	ALUOp0
0	1	1	0	0	0	0	0	0	0	ALUSrcB1
1	1	0	0	0	0	0	0	0	0	ALUSrcB0
0	0	1	0	0	0	1	0	1	0	ALUSrcA
0	0	0	0	1	0	0	1	0	0	RegWrite
0	0	0	0	0	0	0	1	0	0	RegDst



# 多周期控制信号定义: Defined 10+6+? control

## □ 实验十定义的数据通路控制信号

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	请填写信号赋值时 对应操作	
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





# 兼容Exp10的数据通路完善输出信号真值表

状态 \ 输出信号	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
	IF	ID	MEM-Ex	MEM-RD	LW_WB	MEM_W	R_Exc	R_WB	Beq_Exc	J
PCWrite	1	0	0	0	0	0	0	0	0	1
PCWriteCond	0	0	0	0	0	0	0	0	1	0
<b>lorD</b>	0	0	0	1	0	1	0	0	0	0
MemRead	1	0	0	1	0	0	0	0	0	0
MemWrite	0	0	0	0	0	1	0	0	0	0
<b>IRWrite</b>	1	0	0	0	0	0	0	0	0	0
MemtoReg	00	00	00	00	01	00	00	00	00	00
<b>PCSource1</b>	0	0	0	0	0	0	0	0	0	1
<b>PCSource0</b>	0	0	0	0	0	0	0	0	1	0
<b>ALUSrcA</b>	0	0	1	0	0	0	0	0	1	0
ALUSrcB1	0	1	1	0	0	0	0	0	0	0
ALUSrcB0	1	1	0	0	0	0	0	0	0	0
RegWrite	0	0	0	0	1	0	0	1	0	0
RegDst	00	00	00	00	00	00	00	01	00	00
Branch	0	0	0	0	0	0	0	0	1	0
ALUOp1	0	0	0	0	0	0	1	0	0	0
ALUOp0	0	0	0	0	0	0	0	0	1	0
<b>MEM_IO</b>	0	0	0	1	0	1	0	0	0	0

请分析填入



# CPU部件之二-控制器：MCtrl

## □ 多周期控制器

- CPU主要部件之一
- 寄存器传输控制者：
  - 根据状态图将编码转换成命令

## □ 基本功能

- 微操作控制
- 数据传输通道控制

## □ 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

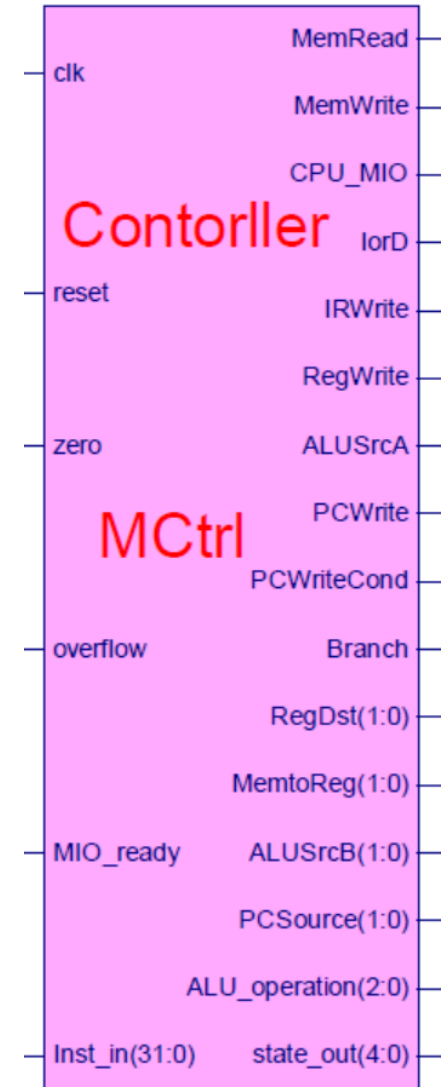


# 控制器接口文档- MCtrl.v

```
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] PCSrc,
              output reg PCWrite,
              output reg PCWriteCond,
              output reg Branch
            );

endmodule
```





# 控制器实现

## □ 控制器实现方案

- 9+条指令实现与单周期方案相同采用二级译码方案
  - 主控制器输出ALUop
  - ALU译码电路调用单周期设计模块

## □ 状态机实现方法

- 状态机实现请参考数字逻辑状态机描述的三种方式
- 状态转换实现选择：
  - 根据状态方程/激励方程：有利于学习理解状态实现原理
  - 根据状态表HDL直接描述：工程设计方便
- 激励方程和输出信号：
  - HDL直接描述：工程设计方便
  - SOP门电路
  - ROM
  - MUX
- 本实验可任取一种状态机方式实现

} 有利于学习理解组合电路原理与结构

# 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, AAAAAAAAAA, BBBBBBBB,
CCCCCCCC, DDDDDDDD, EEEEEEEE, FFFFFFFF, 557EF7E0, D7BDFBD9, D7BDFDB9, DFCFFCFB, DFCFBFFF, F7F3DFFF,
FFFDF3D, FFFF9DB9, FFFBFCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7BDFDB9, 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





# 多周期CPU控制器设计

—控制实验十设计的数据通路





# 设计工程：OExp11-OwnMCPU

## ◎ 设计多周期CPU之控制器

- ☞ 根据Exp10数据通路及指令设计状态图和状态真值表
- ☞ 根据状态表完成控制器电路，HDL描述实现
  - ◎ 必须根据状态表结构描述或激励方程描述实现
- ☞ 仿真测试控制器模块

## ◎ 集成替换验证后的控制器模块

- ☞ 替换实验十(Exp10)中的ctrl.ngc核
- ☞ 顶层模块沿用Exp10：模块名：Top\_OExp10\_OwnMCPU.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
- ☞ 运算控制测试：Function译码测试



# 设计要点

## □ 设计主控制器模块

### ■ 完成输出信号真值表

- 用HDL直接描述实现状态转换并输出控制信号
- 或用激励方程实现状态转换并输出控制信号

## □ 设计ALU操作译码

- 分离出单周期的ALU译码模块并修改调用
- 使用DEMO作功能初步调试
  - ALU必须运算包含“nor”操作
  - 否则需要修改或重新设计调试程序

## □ 仿真主控制器电路模块

- 可以单独或合并仿真，但最后要合并为一个控制模块



# 多周期控制器设计

## 激励方程描述实现



# 激励方程状态机HDL描述结构

## □ 主控制器状态机描述结构

```
parameter IF = 4'b0000,
parameter AND=3'b000, OR=3'b001, ADD=3'b010, SUB=3'b110, .....
```

```
`define Datapath_signals {PCWrite, PCWriteCond, .....
```

```
always @ (posedge clk or posedge reset)
```

```
if (reset==1) Q <= IF;
```

```
else Q <= D;
```

状态转换

```
D3 = Q3nQ2nQ1nQ0n (Op5Op4Op3Op2Op1Op0 + Op5Op4Op3Op2Op1Op0)
D2 = Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0 + Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0 +
      Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0 + Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0
D1 = Q3nQ2nQ1nQ0n (Op5Op4Op3Op2Op1Op0 + Op5Op4Op3Op2Op1Op0) +
      Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0 + Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0
D0 = Q3nQ2nQ1nQ0n + Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0 +
      Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0 + Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0 +
      Q3nQ2nQ1nQ0nOp5Op4Op3Op2Op1Op0
```

激励方程描述

输出变量(信号)描述

数据通路控制

ALU操作译码描述

ALU操作控制



# 激励方程实现状态机参考

## □ 参数定义

### ■ 状态变量

**parameter** IF = 4'b0000, ID = 4'b0001, Mem\_Ex = 4'b0010, Mem\_RD = 4'b0011,  
LW\_WB = 4'b0100, Mem\_W = 4'b0101, R\_Exc = 4'b0110, R\_WB = 4'b0111,  
Beq\_Exc = 4'b1000, J = 4'b1001, Error = 4'b1111;

### ■ 输出变量宏定义

**`define** Datapath\_signals {PCWrite, PCWriteCond, IorD, MemRead, MemWrite, IRWrite,  
MemtoReg, PCSrc, ALUSrcA, ALUSrcB, RegWrite, RegDst, Branch, ALUOp, CPU\_MIO}

### ■ 输出变量值：根据输出信号真值表

**parameter** value0 = 20'b1001010000000100000000, value1 = 20'b00000000000001100000000,  
value2 = 20'b000000000000011000000000, value3 = 20'b0011000000000000000001,  
value4 = 20'b000000000100000010000000, value5 = 20'b0010100000000000000001,  
value6 = 20'b0000000000000100000000100, value7 = 20'b00000000000000001010000,  
value8 = 20'b01000000000110000001010, value9 = 20'b1000000001000000000000;

**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;



$$D_3 = \text{State1 (Beq + Jump)}$$

$$D_2 = \text{State1 Rtype} + \text{State2 Store} + \text{State3 Load} + \text{State6 Rtype}$$

$$D_1 = \text{State1 (Rtype + LS)} + \text{State2 Load} + \text{State6 Rtype}$$

$$D_0 = \text{State0} + \text{State1 Jump} + \text{State2 Load} + \text{State2 Store} + \text{State6 Rtype}$$

## □ 激励方程描述

- 根据D触发器输入方程用**assign**描述
- 也可用逻辑图描述
- 本参考采用状态变量和操作码分别描述然后组合
  - 状态译码描述

**assign** s0 = ~|Q;

//if Q=0000 then s0 = 1

**assign** s1 = ~Q[3] && ~Q[2] && ~Q[1] && Q[0];

//if Q=0001 then s1 = 1

**assign** s2 = ~Q[3] && ~Q[2] && Q[1] && ~Q[0];

//if Q=0010 then s2 = 1

**assign** s3 = ~Q[3] && ~Q[2] && Q[1] && Q[0];

//if Q=0011 then s3 = 1

**assign** s4 =

//if Q=0100 then s4 = 1

**assign** s5 =

//if Q=0101 then s5 = 1

**assign** s6 =

//if Q=0110 then s6 = 1

**assign** s7 =

//if Q=0111 then s7 = 1

**assign** s8 =

//if Q=1000 then s8 = 1

**assign** s9 =

//if Q=1001 then s9 = 1

请完成后继部分



### □ 操作码译码描述

```
assign Rtype = ~|OP; //if OP=000000 then Rtype = 1
assign LS = (OP == 6'b10x011) ? 1 : 0; //if OP=10x011 then LS = 1
assign Beq = (OP == 6'b000100) ? 1 : 0; //if OP=000100 then Ibeq = 1
assign Jump = (OP == 6'b000010) ? 1 : 0; //if OP=000010 then Jump = 1
assign Load = (OP == 6'b100011) ? 1 : 0; //if OP=100011 then Load = 1
assign Store = (OP == 6'b101011) ? 1 : 0; //if OP=101011 then Store = 1
```

请完成后继部分

### □ 激励方程合成描述

```
assign D[3] = D3 = State1 (Beq + Jump)
assign D[2] = D2 = State1 Rtype + State2 Store + State3 Load + State6 Rtype
assign D[1] = D1 = State1 (Rtype + LS) + State2 Load + State6 Rtype
assign D[0] = D0 = State0 + State1 Jump + State2 Load + State2 Store + State6 Rtype
```

### □ 状态转换描述

```
always @ (posedge clk or posedge reset)
    if (reset==1) Q <= IF;
    else Q <= D;
```





## □ 输出变量(信号)描述

```
always @ * begin
```

```
case(Q)
```

```
//state
```

```
IF:      `Datapath_signals = value0;
```

```
ID:      `Datapath_signals = value1;
```

```
Mem_Ex:  `Datapath_signals = value2;
```

```
Mem_RD:  `Datapath_signals = value3;
```

```
LW_WB:   `Datapath_signals = value4;
```

```
Mem_W:   `Datapath_signals = value5;
```

```
R_Exc:   `Datapath_signals = value6;
```

```
R_WB:    `Datapath_signals = value7;
```

```
Beq_Exc: `Datapath_signals = value8;
```

```
J:       `Datapath_signals = value9;
```

```
default: `Datapath_signals = value0;
```

```
endcase
```

请完成后继部分

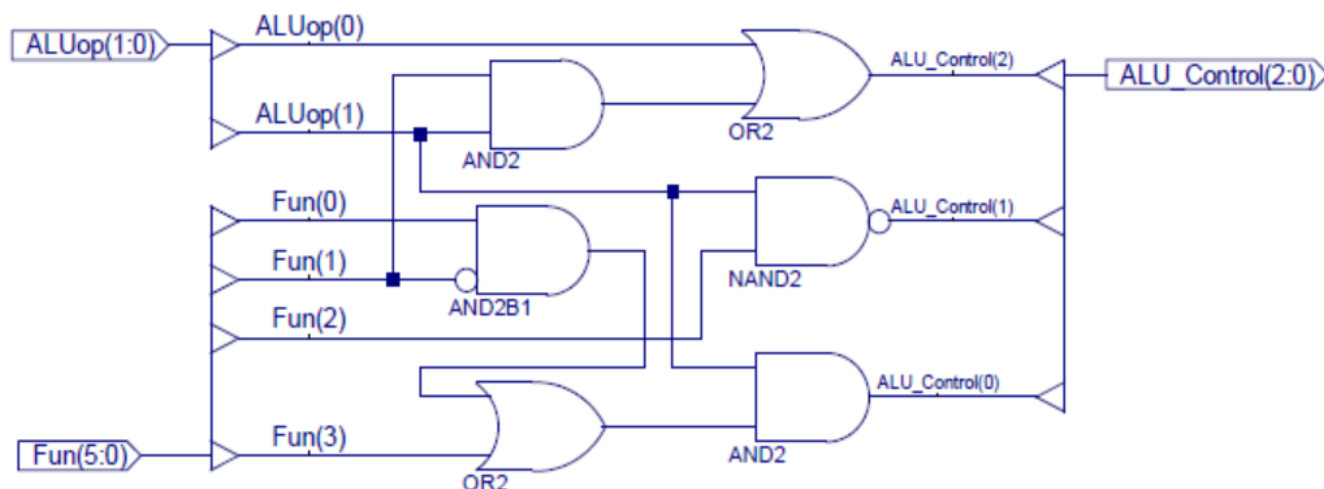
# ALU操作译码

## □ 本实验采用二级译码

- 调用单周期ALU译码电路
- 修改单周期控制器模块

□ 分离出ALU译码部分(原理图2部分)

**ALU\_Decoder**     **ALU\_D(.ALUop(ALUop),**  
                          **.Fun(Inst\_in[5:0]),**  
                          **.ALU\_Control(ALU\_operation));**





# 多周期控制器设计

## HDL直接描述实现



# 控制器HDL直接描述结构

## □ 主控制器状态机描述结构

```
parameter IF = 4'b0000,
parameter AND=3'b000, OR=3'b001, ADD=3'b010, SUB=3'b110, .....
```

```
`define Datapath_signals {PCWrite, PCWriteCond, .....
```

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

状态机

.....

```
end
```

```
always @* begin
```

数据通路控制

.....输出变量(信号)描述

```
end
```

ALU操作译码描述

ALU操作控制



# 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 <= R_Exc;           //R-type OP
            6'b100011: state <= Mem_Exc;         //Lw
            .....
            6'b000100: state <= Beq_Exc;         //Beq
          default:    state <= Error;
        endcase
      Mem_Exc:begin
                .....
      Error:    state <= Error;
    default:   state <= Error;
  endcase
```



## □ 输出变量(信号)描述: 与激励方程时相同

```
always @ * begin
```

```
case(Q)
```

```
//state
```

```
IF:      `Datapath_signals = value0;
```

```
ID:      `Datapath_signals = value1;
```

```
Mem_Ex:  `Datapath_signals = value2;
```

```
Mem_RD:  `Datapath_signals = value3;
```

```
LW_WB:   `Datapath_signals = value4;
```

```
Mem_W:   `Datapath_signals = value5;
```

```
R_Exc:   `Datapath_signals = value6;
```

```
R_WB:    `Datapath_signals = value7;
```

```
Beq_Exc: `Datapath_signals = value8;
```

```
J:       `Datapath_signals = value9;
```

```
default: `Datapath_signals = value0;
```

```
endcase
```

请完成后继部分



# ALU操作译码

```
always @ * begin
    case(ALUOp)
        2'b00: ALU_operation = 3'b010;           //add计算地址
        2'b01: ALU_operation = 3'b110;           //sub比较条件
        2'b10:
            case (Inst_in[5:0])
                6'b100000: ALU_operation = ADD;
                6'b100010: ALU_operation = SUB;
                6'b100100: ALU_operation = AND;
                6'b100101: ALU_operation = OR;
                6'b100111: ALU_operation = NOR;
                6'b101010: ALU_operation = SLT;
                6'b000010: ALU_operation = SRL;       //shfit 1bit right
                6'b000000: ALU_operation = XOR;
                default: ALU_operation = ADD;
            endcase
        2'b11: ALU_operation = 3'b111;           //slti
    endcase
end
```



# 多周期控制器实现

## 集成替换



# 控制器集成替换

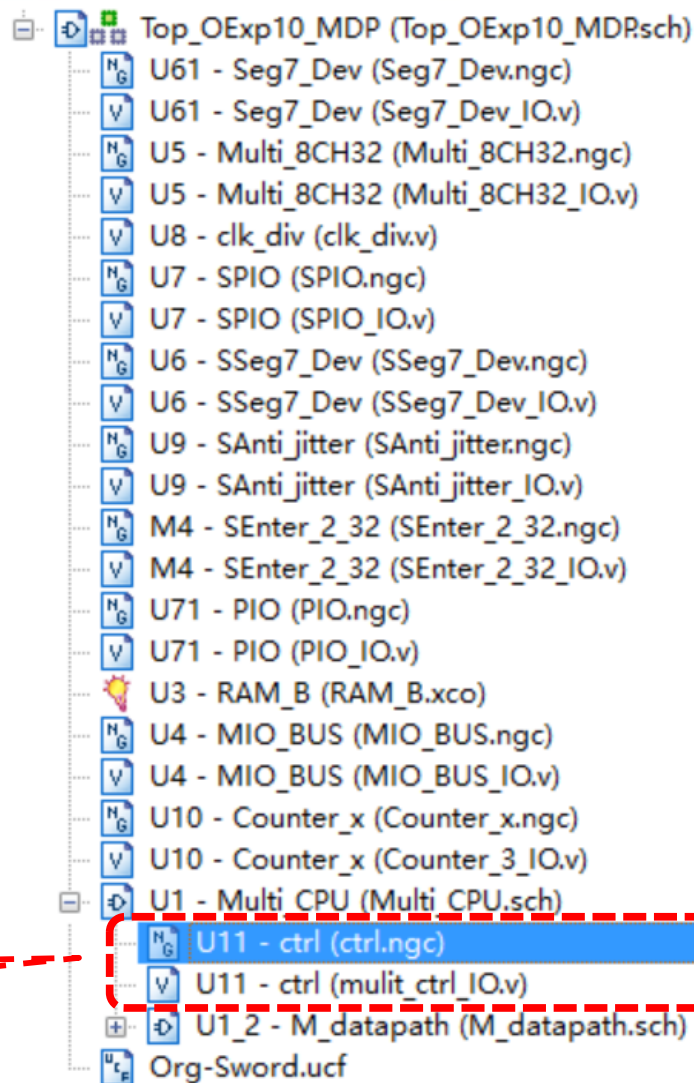
## □ 集成替换

- 仿真正确后替换Exp10的控制器IP核

## □ 移除工程中的控制器核

- Exp10工程中移控制器核关联
- 删除工程中控制器核文件
  - ctrl.ngc文件
  - 在Project菜单中运行:  
**Cleanup Project Files ...**
- 建议用Exp10资源重建工程
  - 除ctrl.ngc核

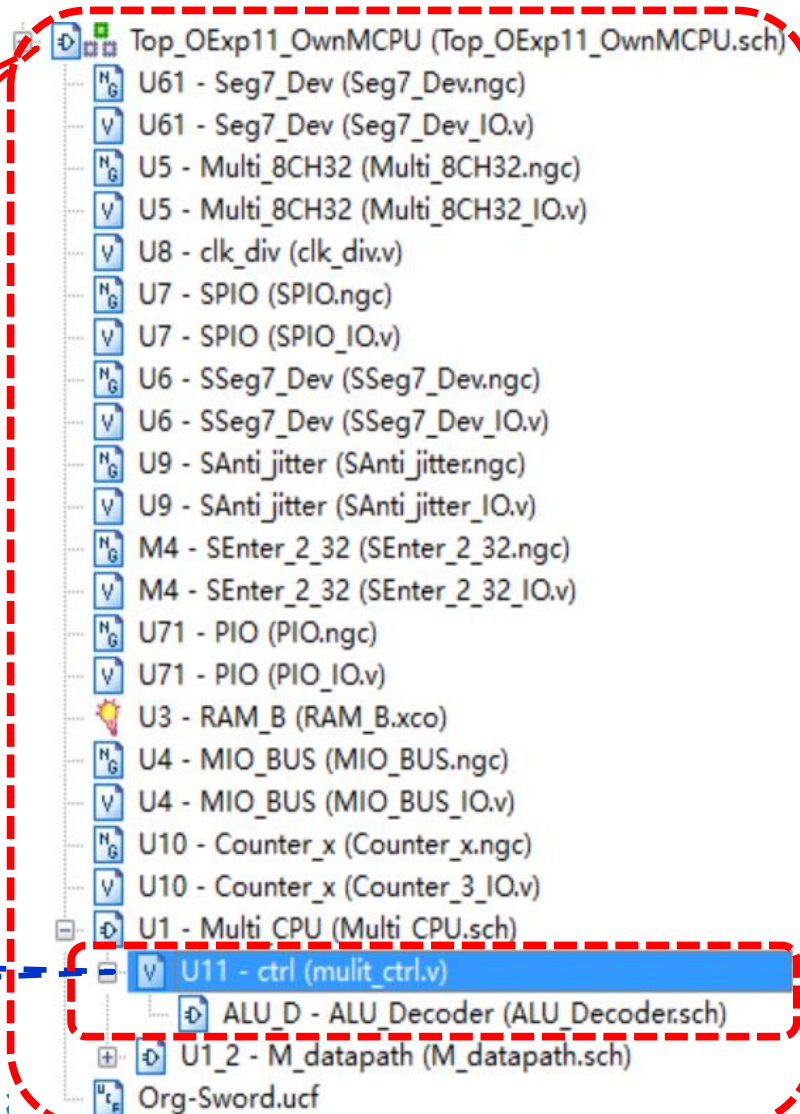
Exp10需要清理的核



## 集成替换ctrl核后的模块层次结构

Exp11完成数据通路替换后的模块调用关系

替换后的控制器模块



## □ 使用DEMO程序目测控制器功能(同实验十)

### ■ DEMO接口功能

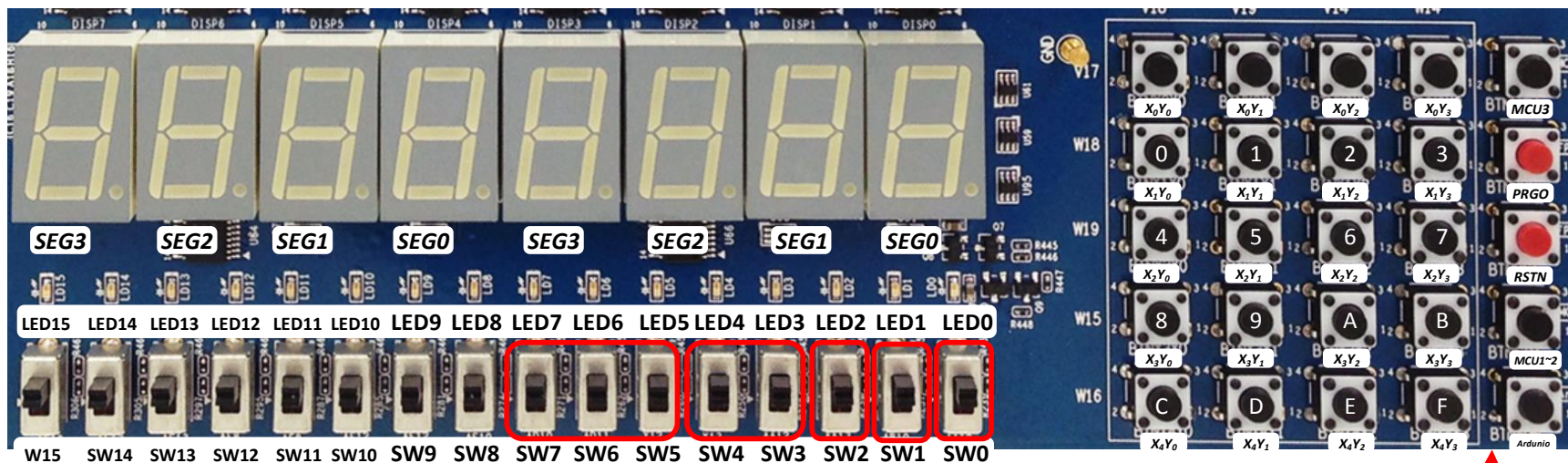
#### □ SW[7:5]=000, SW[2]=0(全速运行)

- 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

SW[2]=CPU单步时钟选择

没有使用

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指令

## □ 设计ALU指令测试程序替换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,r3;           //r4=00000004
        add r5,r4,r2;           //r5=00000005
        add r6,r5,r5;           //r6=0000000A
        nor r7,r5,r5;           //r7=FFFFFFFA
        sub r8,r7,r5;           //r8=FFFFFFF5
        and r9,r8,r5;           //r9=00000005
        and r10,r8,r6;          //r10=00000000
        or r11,r5,r6;           //r11=0000000F
        or r12,r11,r7;          //r12=0000000A
        slt r13,r5,r7;          //r13=00000000
        .....
j loop;
```



# 测试程序参考： LW/SW

## □ 设计LW/SW程序替换DEMO程序

- 参考Lab5通道测试设计。测试结果通过CPU输出信号单步观察
- 存储器地址通过Addr\_out通道4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 test_sw;                  //循环测试
      j    start;                          //循环测试。立即数通道: 00000014
test_sw: .....                            //增加写SW测试, 如14和48单元交换
      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地址计算、存储单元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



# 动态LW/SW测试

## □ 利用七段显示设备可以设计动态测试程序

- 7段码显示器的地址是E0000000/FFFFFFE0
- LED显示地址是F0000000/FFFFFF00
- SW指令输出测试结果: sw
- 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初始化数据同Exp10

00A52820, AC650000, 8C650000, 00A85824, 01A26820, 11A00017, 8C650000, 01CE9020,  
0252B020, 02569020, .....

代码区

F0000000, 000002AB, 80000000, 0000003F, 00000001, FFF70000, 0000FFFF, 80000000,  
00000000, 11111111, 22222222, 33333333, 44444444, 55555555, 66666666, 77777777,  
88888888, 99999999, aaaaaaaaa, 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

- ALU指令测试结果记录
  - 自行设计记录表格
- LW/SW指令测试结果记录
  - 自行设计记录表格
- 动态存储模块测试记录
  - 自行设计记录表格



# 思考题



- 用HDL直接描述状态机时同时输出控制信号需要如何修改？
- 设计bne指令需要增加控制信号吗？
- 扩展下列指令，控制器将作如何修改：
  - R-Type: srl\*, jr, jalr, eret\*;
  - I-Type: addi, andi, ori, xori, lui, bne, slti
  - J-Type: Jal;
- 此时用二级译码有优势吗？
- 状态机调试你有什么建议？



● END