#### Computer Organization & Design实验与课程设计

# Lab04-2 CPU设计—控制器

Ma De (马德)

made@zju.edu.cn

2022

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. 运用寄存器传输控制技术
- 2. 掌握CPU的核心: 指令执行过程与控制流关系
- 3. 设计控制器
- 4. 学习测试方案的设计
- 5. 学习测试程序的设计

# 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. Sword 2.0/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 RV32I的指令特点,了解控制器的原理,设计并测试控制器
- 任务一·: 用硬件描述语言设计实现控制器
  - □根据Exp04-1数据通路及指令编码完成控制信号真值表
  - □此实验在Exp04-1的基础上完成,替换Exp04-1的控制器核

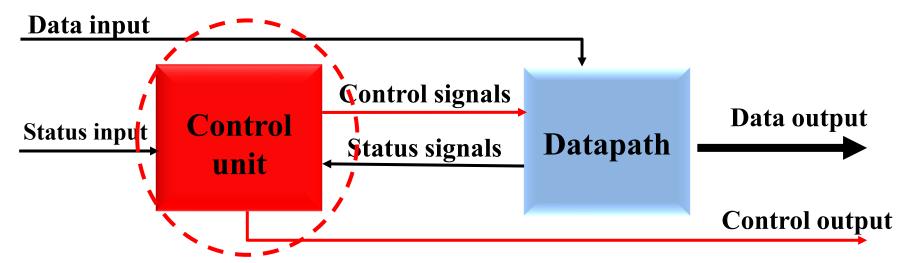
- 任务二·: 设计控制器测试方案并完成测试
  - □OP译码测试: R-格式、访存指令、分支指令,转移指令
  - □运算控制测试: Function译码测试

# RISC-V RV32I控制器的原理介绍

#### **CPU** organization

#### □ Digital circuit

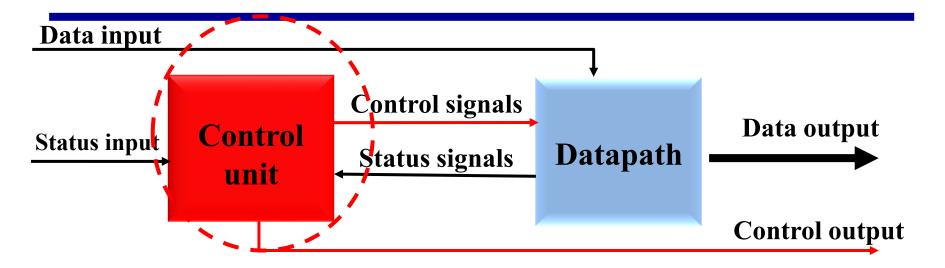
General circuits that controls logical event with logical gates Hardware



#### **□** Computer organization

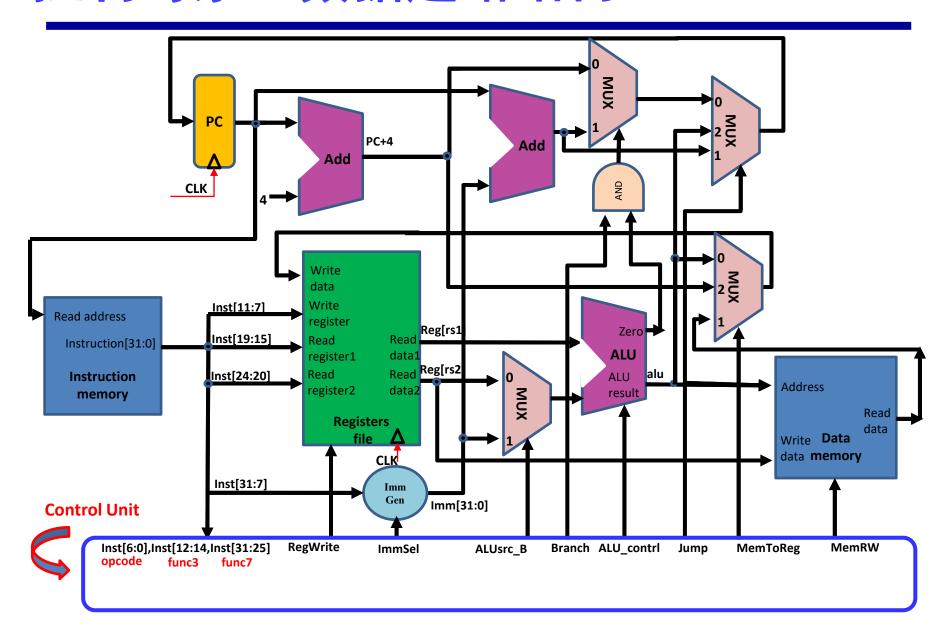
Special circuits that processes logical action with instructions
 Software

#### Control unit

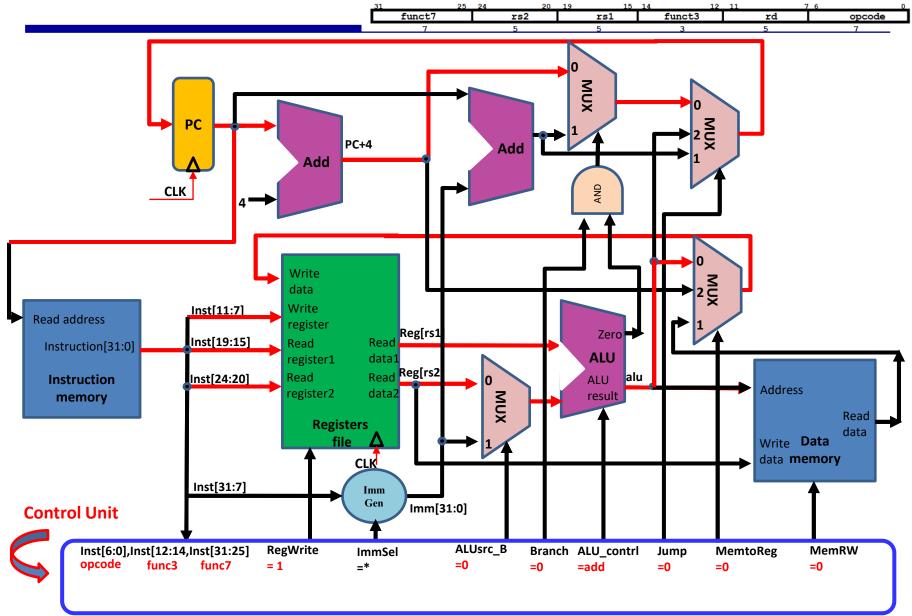


- □控制单元
- □ 控制单元作为处理器的一部分,用以告诉数据通路需要做什么。包含了取指单元(PC及地址计算单元)、译码单元、控制单元(时序信号形成及微操作控制信号形成电路)、中断等

# 控制对象: 数据通路结构

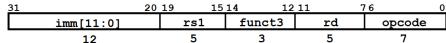


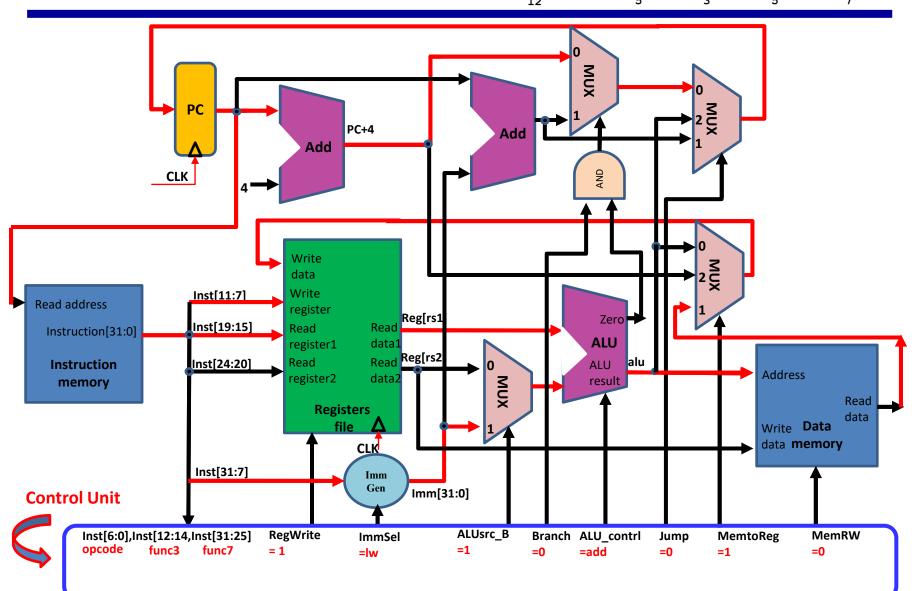
#### Control unit---add



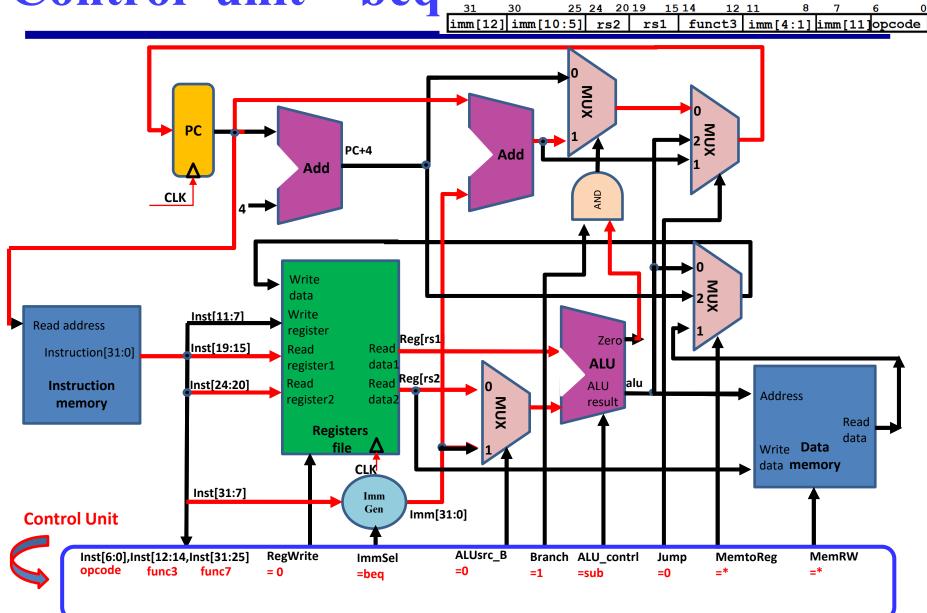
#### Control unit---sw rs2 imm[4:0] Imm[11:5] rs1 funct3 opcode offset[11:5] base width offset[4:0 STORE src **MUX** Add PC+4 Add AND CLK Write data Write Inst[11:7] Read address register Reg[rs1 Zero Inst[19:15] Read Read Instruction[31:0] **ALU** register1 data1 Read Reg[rs2 Read alu Inst[24:20] Instruction ALU Address register2 data2 MUX result memory Read **Registers** data Write Data data **memory** CLK Inst[31:7] Imm Gen Imm[31:0] **Control Unit** ALUsrc B Inst[6:0],Inst[12:14,Inst[31:25] RegWrite ALU contrl MemtoReg MemRW **ImmSel Branch** Jump opcode func3 func7 = 0=1 =0 =add =0 =\* =1 =sw

#### Control unit----lw

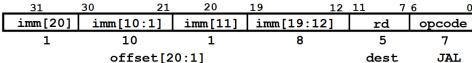


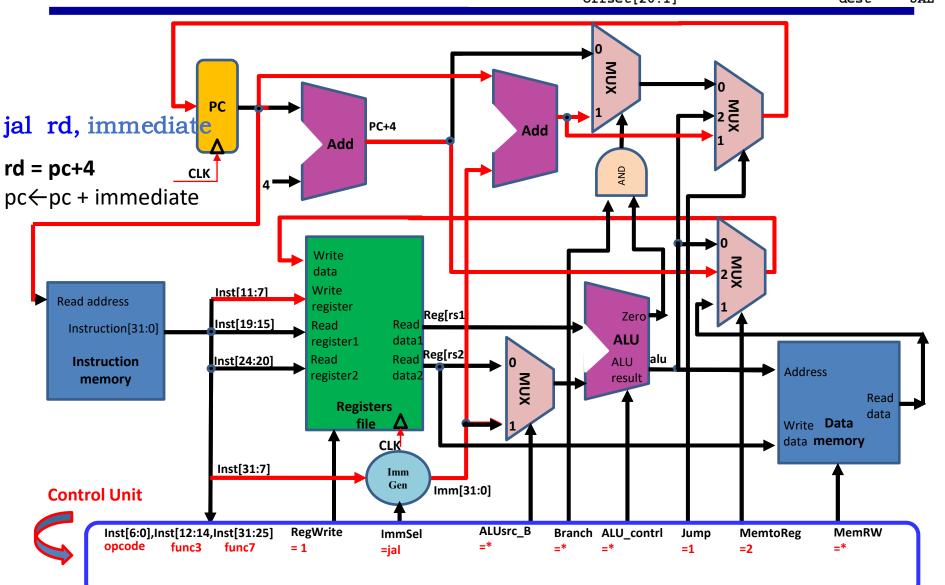


### Control unit---beq









# 控制信号定义

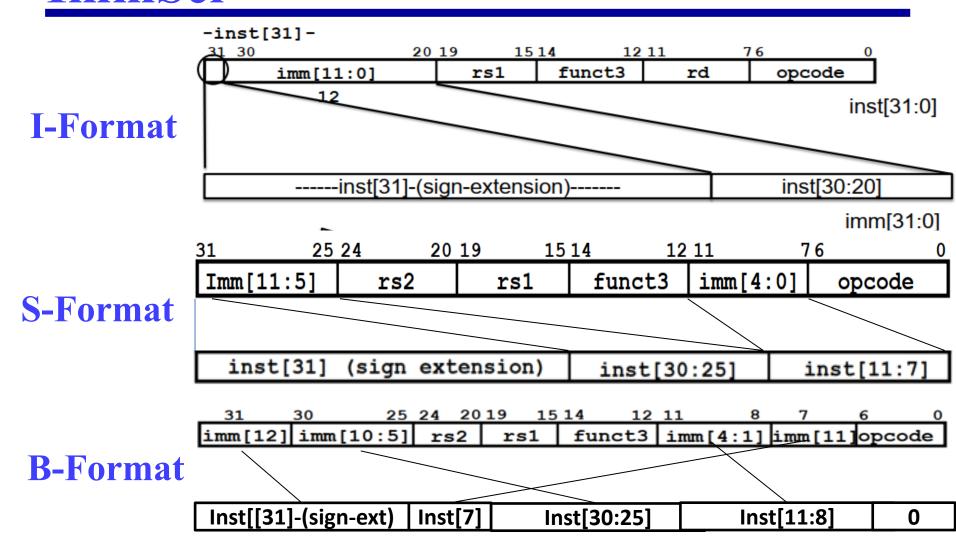
#### □通路与操作控制

信号	源数目	功能定义	赋值0时动作	赋值1时动作	赋值2时动作
ALUSrc_B	2	ALU端口B输入选择	选择源操作数寄存 器2数据	选择 <b>32</b> 位立即数(符 号扩展后)	-
MemToReg	3	寄存器写入数据选择	选择ALU输出	选择存储器数据	选择PC+4
Branch	2	Beq指令目标地址选择	选择PC+4地址	选择转移目的地址 PC+imm(zero=1)	-
Jump	3	J指令目标地址选择	由Branch决定输出	选择跳转目标地址 PC+imm(JAL)	-
RegWrite	-	寄存器写控制	禁止寄存器写	使能寄存器写	-
MemRW	-	存储器读写控制	存储器 <b>读使能</b> , 存储器写禁止	存储器 <b>写使能</b> , 存储器读禁止	-
ALU_Control	000-111	3位ALU操作控制	参考表ALU_Control		
ImmSel 2022/3/2	000-111	3位立即数组合控制	参考表ImmSel		apter 6 15

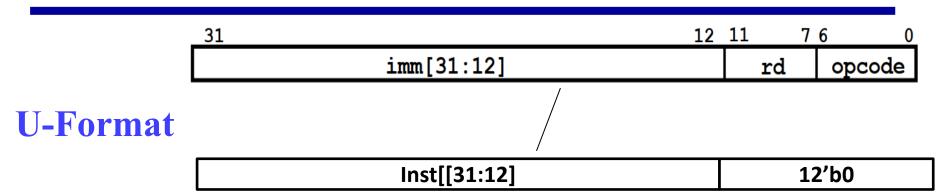
# 主控制器信号真值表

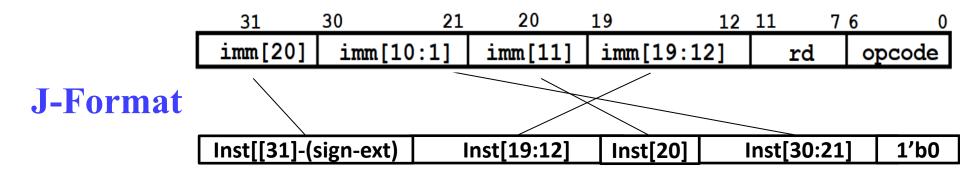
Inst[31:0]	Banch	Jump	ImmSel	ALUSrc_B	ALU_Control	MemRW	RegWrite	MemtoReg
add	0	0	*	Reg	Add	Read	1	ALU
sub	0	0	*	Reg	Sub	Read	1	ALU
(R-R Op)	0	0	*	Reg	(Op)	Read	1	ALU
addi	0	0	I	lmm	Add	Read	1	ALU
lw	0	0	I	lmm	Add	Read	1	Mem
SW	0	0	S	lmm	Add	Write	0	*
beq	0	0	В	Reg	Sub	Read	0	*
beq	1	0	В	Reg	sub	Read	0	*
jal	0	1	J	lmm	*	Read	1	PC+4
lui	0	0	U	lmm	Add	Read	1	ALU

#### **ImmSel**



#### **ImmSel**





### **ImmSel**

Instruction type	Instruction opcode[6:0]	Instruction operation	(sign-extend)immediate	Imm Sel
	0000011	Lw;lbu;lh; lb;lhu		
I-type	0010011	Addi;slti;slti u;xori;ori;a ndi;	(sign-extend) instr[31:20]	00
	1100111	1 jalr		
S-type	0100011	Sw;sb;sh	(sign-extend) instr[31:25],[11:7]	01
B-type	1100011	Beq;bne;blt ;bge;bltu;b geu	(sign-extend) instr[31],[7],[30:25],[11:8], 1'b0	10
J-type	1101111	jal	(sign-extend) instr[31],[19:12],[20],[30:21],1 'b0	11
2022/3/2			Chapter	6 19

	inst[30]	LUI	0110111	rd			imm[31:12]	
		AUIPC	0010111	rd			imm[31:12]	
1	0000000	JAL _	1101111	rd		9:12]	n[20 10:1 11 1	
sham	0000000	JALR _	1100111	rd	000	rs1	)]	imm[11:
sham	0000000	BEQ	1100011	imm[4:1 11]	000	rs1	rs2	imm[12 10:5]
sham	0100000	BNE	1100011	imm[4:1 11]	001	rs1	rs2	imm[12 10:5]
rs2	0000000	BLT	1100011	imm[4:1 11]	100	rs1	rs2	imm[12 10:5]
rs2	0100000	BGE -	1100011	imm[4:1 11]	101	rs1	rs2	imm[12 10:5]
		BLTU -	1100011	imm[4:1 11]	110	rs1	rs2	imm[12 10:5]
rs2	0000000	BGEU_	1100011	imm[4:1 11]	111	rs1	rs2	imm[12 10:5]
rs2	0000000	LB	0000011	rd	000	rs1	)]	imm[11:
rs2	0000000	LH	0000011	rd	001	rs1	)]	imm[11:
rs2	0000000	LW	0000011	rd	010	rs1	)]	imm[11:
rs2	0000000	LBU	0000011	rd	100	rs1		imm[11:
rs2	0100000	LHU -	0000011	rd	101	rs1		imm[11:
		SB _	0100011	imm[4:0]	000	rs1	rs2	imm[11:5]
rs2	0000000	SH _	0100011	imm[4:0]	001	rs1	rs2	imm[11:5]
rs2	0000000	SW	0100011	imm[4:0]	010	rsl	rs2	imm[11:5]
		ADDI	0010011	rd	000	rs1	,	imm[11:
		SLTI	0010011	rd	010	rs1	)]	imm[11:
		SLTIU	0010011	rd	011	rs1	)]	imm[11:
		XORI	0010011	rd	100	rs1	1	imm[11:
		ORI	0010011	rd	110	rs1	,	imm[11:
nred	0000	ANDI	0010011	rd	111	re1	)]	imm[11:

• •			-	-		
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	$^{\mathrm{rd}}$	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT

rs1

rs1

rs1

rs1

rs1

inst[14:12]

011

100

101

101

110

rd

rd

rd

rd

rd

rd

inst[6:2]

0110011

0110011

0110011

0110011

0110011

0110011

SLTU

XOR

SRL

SRA

OR

AND

Ш	0000	pred	succ	00000	000	00000	0001111	I fence
Ш	0000	0000	0000	00000	001	00000	0001111	I fence.i
Ш	00000000000			00000	000	00000	1110011	I ecall
Ш	00000000001			00000	000	00000	1110011	I ebreak
Ш		csr		rs1	001	rd	1110011	I csrrw
	csr			rs1	010	rd	1110011	I csrrs
		csr		rs1	011	rd	1110011	I csrrc
Ш	csr			zimm	101	rd	1110011	I csrrwi
$\ $	csr			zimm	110	rd	1110011	I cssrrsi
$\ [$		csr		zimm	111	rd	1110011	I csrrci

Instruction type encoded using only 9 bits inst[30],inst[14:12], inst[6:2]

特殊指令本实验暂不做考虑

Instruction opcode	op	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
		add	000	0000000		add	010
		sub	000	0100000	10	sub	110
		sll	001	0000000		sll	-
		slt	010	0000000		slt	111
R-type	0110011	sltu	011	0000000		sltu	-
		xor	100	0000000		xor	011
		srl	101	0000000		srl	101
		sra	101	0100000		sra	-
		or	110	000000		or	001
		and	111	0000000		and	000

Instruction opcode	ор	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
S-Type		sb	000	-		add	010
	0100011	sh	001	-	00	add	010
		sw	010	-		add	010
Instruction opcode	ор	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
		Beq	000	-		sub	110
		· ·					
		Bne	001	-		sub	110
P. Typo	1100011	Bne	001	-	01	sub	110
В-Туре	1100011	Bne	001	-	01	sub	110
B-Type	1100011	Bne	001	-	01	sub	110

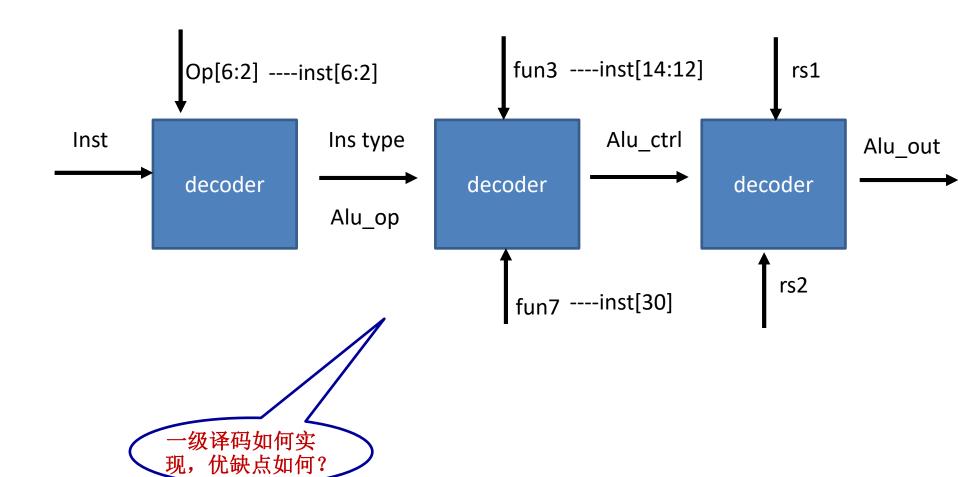
2022/3/2

Instruction opcode	ор	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
U-Type	0110111	lui	1	1			-
	0010111	auipc	-	-	_	-	-
Instruction opcode	ор	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
J-Type	1101111	jal	-	-	-	-	-

Instruction opcode	ор	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
	0000011	Lb	-	-		add	010
		Lh	-	-	00	add	010
I Turno		Lw	010	-		add	010
I-Type		Lbu	1	-		add	010
		lhu	1	1		add	010

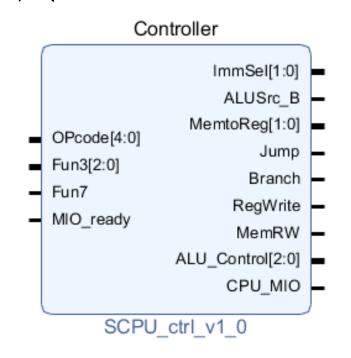
Instruction opcode	ор	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
		addi	000	-		add	010
		slti	010	-	11	slt	111
		sltiu	011	-		sltu	-
		xori	100	-		xor	011
I-Type	0010011	ori	110	-		or	001
		andi	111	-		and	000
		slli	001	0000000		sll	-
		srli	101	0000000	_	srl	101
		srai	101	0100000		sra	-

# ALU操作译码----多级译码



### CPU部件之控制器接口: SCPU ctrl

- □ SCPU\_ctrl
  - CPU主要部件之一
  - 寄存器传输控制者:编码转换成命令
- □基本功能
  - ■微控制控制
  - 数据传输通道控制
  - 时序控制: 单周期时序在那里?
- □接口要求-SCPU\_ctrl
  - ■控制器接口信号如右图



# 控制器接口信号标准- SCPU ctrl.v

```
SCPU ctrl(input[4:0]OPcode, //Opcode----inst[6:2]
module
                input Fun7, //Function----inst[30]
                input MIO ready, //CPU Wait
                output reg [1:0]ImmSel, //立即数选择控制
                output reg ALUSrc B, //源操作数2选择
                output reg MemtoReg, //写回数据选择控制
                output reg Jump, //jal
                output reg Branch, //beq
                output reg RegWrite, //寄存器写使能
                output reg MemRW, //存储器读写使能
                output reg [2:0]ALU Control, //alu控制
                output reg CPU MIO //not use
                );
```

#### endmodule

■任务一:用硬件描述语言设计实现控制器

□根据Exp04-1数据通路及指令编码完成控制信号真值表

□此实验在Exp04-1的基础上完成,替换Exp04-1的控制器核

# CPU之控制器设计

-控制04-1设计的数据通路

# 设计工程: OExp04-SCPU\_ctrl

#### ◎设计CPU之控制器

- 每根据理论课分析讨论设计数据通路的的控制器
- € 采用HDL描述
- € 仿真测试控制器模块

#### ◎集成替换验证通过的数据通路模块

- € 替换 (Exp04-1)中的SCPU ctrl核
- € 顶层模块延用Exp04

#### ◎测试控制器模块

- € 设计测试程序(RISCV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

2022/3/2 Chapter 6

31

# 设计要点

#### □设计主控制器模块

- ■写出控制器的函数表达式
- 结构描述实现电路
- □设计ALU操作译码
  - ■写出ALU操作译码函数表达式
  - 结构描述实现电路
  - ■使用DEMO作功能初步调试
- □仿真二个控制器电路模块
  - 可以单独或合并仿真,但最后要合并为一个控制模块

2022/3/2 Chapter 6

32

# 主控制器HDL描述结构

#### □指令译码器参考描述

```
`define CPU ctrl signals {ALUSrc B,MemtoReg,RegWR,MemWrite,Branch,Jump,ALUop}
        always @* begin
                  case(OPcode)
                  5'b01100: begin CPU_ctrl_signals = ?; end
                                                               //ALU
                  5'b00000: begin CPU ctrl signals = ?; end
                                                               //load
                  5'b01000: begin CPU ctrl signals = ?; end
                                                               //store
                  5'b11000: begin CPU ctrl signals = ?; end
                                                               //beq
                  5'b11011: begin CPU ctrl_signals = ?; end
                                                               //jump
                  5'b00100: begin CPU ctrl signals = ?; end
                                                               //ALU(addi;;;;)
                            begin CPU ctrl signals = ?; end
                  default:
                  endcase
        end
```

某些指令仅需部分控制信号, 无关信号可赋定值,避免组合 逻辑赋值不完整而生成latch

33

# ALU操作译码器HDL描述结构

#### □ ALU控制器参考描述

```
assign Fun = \{Fun3, Fun7\};
always @* begin
  case(ALUop)
  2'b00: ALU Control = ?;
                                            //add计算地址
  2'b01: ALU Control = ?;
                                            //sub比较条件
  2'b10:
     case(Fun)
        4'b0000: ALU Control = 3'b010;
                                            //add
        4'b0001: ALU Control = ?; //sub
        4'b1110: ALU Control = ?; //and
        4'b1100: ALU Control = ?; //or
        4'b0100: ALU Control = ?; //slt
        4'b1010: ALU Control = ?; //srl
        4'b1000: ALU Control = ?; //xor
         default: ALU Control=3'bx;
         endcase
  2'b11:
     case(Fun3)
  endcase
```

# 控制器仿真激励代码参考

```
initial begin
        // Initialize Inputs
        OPcode = 0:
        Fun = 0:
        MIO ready = 0;
        #40:
        // Wait 40 ns for global reset to finish。以上是测试模板代码。
        // Add stimulus here
        //检查输出信号和关键信号输出是否满足真值表
        OPcode = 5'b01100; //ALU指令,检查ALUop=2'b10; RegWrite=1
        Fun3 = 3'b000; Fun7 = 1'b0; //add, 检查ALU Control=3'b010
        #20:
        Fun3 = 3'b000; Fun7 = 1'b1;//sub,检查ALU Control=3'b110
        #20:
        Fun3 = 3'b111; Fun7 = 1'b0;//and,检查ALU Control=3'b000
        #20:
        Fun3 = 3'b110; Fun7 = 1'b0;//or,检查ALU Control=3'b001
        #20:
        Fun3 = 3'b010; Fun7 = 1'b0;//slt,检查ALU Control=3'b111
        #20;
```

# 控制器仿真激励代码参考

```
#20;
Fun3 = 3'b101; Fun7 = 1'b0 //srl,检查ALU Control=3'b101
#20:
Fun3 = 3'b100; Fun7 = 1'b0 //xor,检查ALU Control=3'b011
#20:
Fun3 = 3'b111; Fun7 = 1'b1; //间隔
#1;
OPcode = 5'b000000;
                        //load指令,检查ALUop=2'b00,
                        //ALUSrc\ B=1, MemtoReg=1, RegWrite=1
#20:
OPcode = 5'b01000;
                 //store指令,检查ALUop=2'b00, MemRW=1, ALUSrc B=1
#20:
OPcode = 5'b11000;//beg指令,检查ALUop=2'b01, Branch=1
#20;
OPcode = 5'b11011;
                       //jump指令,检查Jump=1
                       //ˈl指令,检查ALUop=2'b11; RegWrite=1
OPcode = 5'b00100;
#20;
                       //addi, 检查ALU Control=3'b010
Fun3 = 3'b000:
#20;
OPcode = 5'hlf;
Fun3 = 3'b000; Fun7 = 1'b0;//间隔
```

end

# 控制器模块时序仿真参考

			41-31	7 1/7						
Name	Value	0 ns	50 ns	100 ns	150 ns	200 ns	250 ns	300 ns	350 ns	400 ns
> M OPcode[4:0]	11111	00000		01100		00 01 11	11011		00100	
> 🛂 Fun3[2:0]	000	000	(111	110 010 10	1 100	111		000 111 11	0 010 101	100
™ Fun7	0									
™ MIO_ready	0									
> = ImmSel[1:0]	00			00		X 01 X 1	11		00	
<sup>™</sup> ALUSrc_B	0									
> 🌉 MemtoReg[1:0]	00	01		00		01 00	10		00	
⅓ Jump	0									
₩ Branch	0									
<sup>™</sup> RegWrite	0									
<b>¼</b> MemRW	0									
> 🌃 ALU_Control[2:0]	010	010	110 000	001 111 10	1 011 XXX	010 11	0 010	X 000 X 00	1 111 101	011
TE COLL MIC	n									
Name	Value	200 ns	220 ns	240 ns	260 ns	280 ns	300 ns	320 ns	340 ns	360 ns
> 3 OPcode[4:0]	11111	00000	01000	11000			/			
► ■ Eup2[2:0]		4	/ 01000		110	)11	<u> </u>	00100		
> 🐝 Fun3[2:0]	000	2 3333	/ 01000	111	110	11	000	00100 111	110	010
> = Fun3[2:0]	000 0		/ 01000		110	11	000		110	010
			,		110	11	000		110	010
⅓ Fun7	0	00	01			011	000		( 110	010
15 Fun7	0			111			000	111	110	(010
I‰ Fun7 I‰ MIO_ready > ■ ImmSel[1:0]	0 0 00		01	111	1		000	111	110	010
I‰ Fun7 I‰ MIO_ready  > ■ ImmSel[1:0] I‰ ALUSrc_B	0 0 00 0	00	01	111	1		000	00	( 110	010
I‰ Fun7 I‰ MIO_ready > ImmSel[1:0] I‰ ALUSrc_B > ImmSel[1:0] > ImmSel[1:0]	0 0 00 0	00	01	111	1		000	00	( 110	(010
Fun7  MIO_ready  MIMMSel[1:0]  ALUSrc_B  MemtoReg[1:0]  Jump	0 0 00 0 0	00	01	111	1		000	00	( 110	(010
Fun7  MIO_ready  MIO_ready  MIMMSel[1:0]  ALUSrc_B  MIMMSel[1:0]  Jump  Jump  Branch	0 0 00 0 0 00 0	00	01	111	1		000	00	110	010
Fun7  MIO_ready  MIO_ready  MIMSel[1:0]  ALUSrc_B  MemtoReg[1:0]  Jump  Branch  RegWrite	0 0 00 0 0 0 0	00	01	111	1		000	00	001	(111

#### 控制器集成替换、CPU集成替换

- □控制器集成替换
  - 仿真正确后替换Exp04-1的控制器IP核
- □ CPU集成替换
  - 仿真正确后替换SOC系统中的CPU
- □集成和替换方法与1ab04-1相同

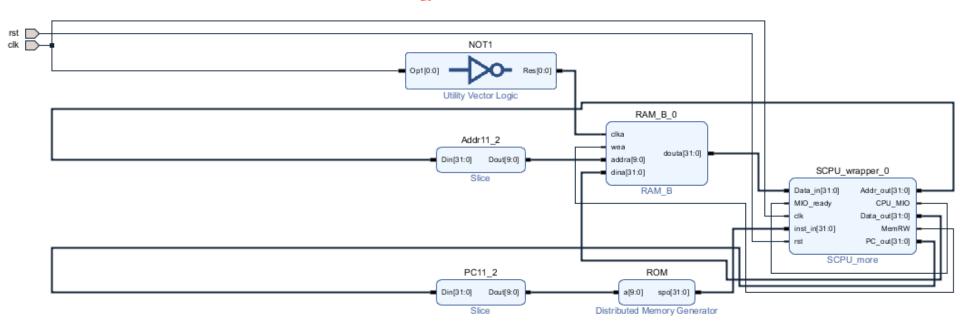


- - W U1: SCPU\_ctrl (SCPU\_ctrl.v)
  - W U2: DataPath (DataPath.v) (11)
    - ALU\_U: ALU (Alu.v)
    - AND2: AND (AND.v)
    - ImmGen\_U: ImmGen (ImmGen.v)
    - MUX0: MUX2T1\_32 (MUX2T1\_32.v)
    - MUX1: MUX2T1\_32 (MUX2T1\_32.v)
    - MUX2: MUX2T1\_32 (MUX2T1\_32.v)
    - MUX3: MUX4T1\_32 (MUX4T1\_32.v)
    - PC: REG32 (REG32.v)
    - Regs\_U: Regs (Regs.v)
    - add32\_0 : add\_32 (add\_32.v)
    - add32\_1: add\_32 (add\_32.v)

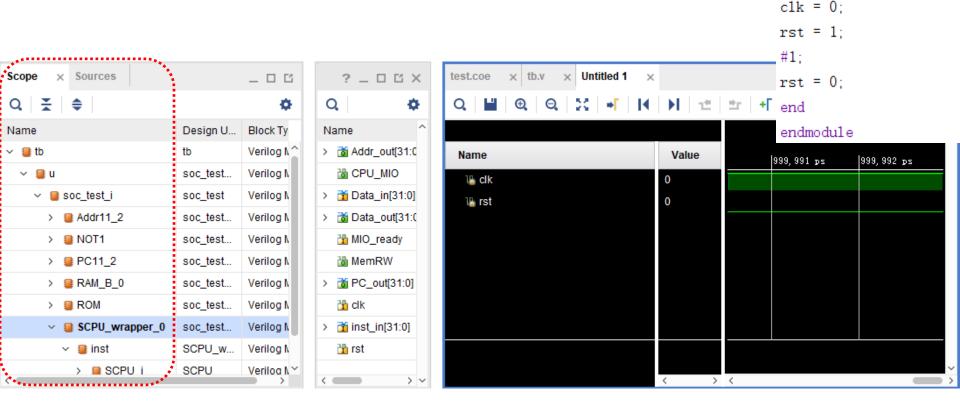
2022/3/2

#### □ CPU设计并集成后,应进行功能仿真·

■ 利用CPU(本实验设计)、RAM、ROM可以搭建 为包含源文件形式,如下图的仿真平台,初始化RAM的数据,初始化ROM的 不能包含EDF文件 指令(可自行选择设计),可以完成CPU各种类型指令的执行,查看CPU数据的输入和输出,以及底层的包括寄存器在内的个模块的状态。(参考原理图进行verilog描述)



- □ 仿真平台搭建完成,编写Testbench开始仿真
  - 由于外部端口只有clk,rst,需要在Scope窗口添加观察信号



module tb():

soc\_test\_wrapper u(

always #5 clk = ~clk:

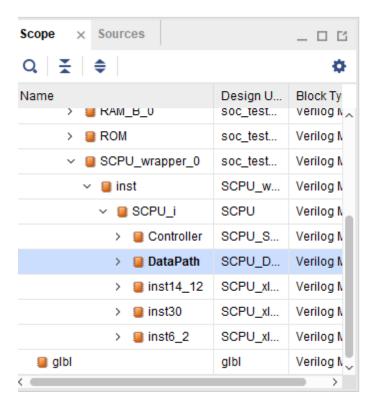
reg clk; reg rst:

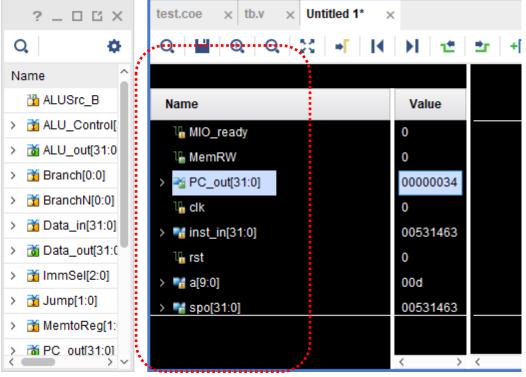
.clk(clk),

.rst(rst)

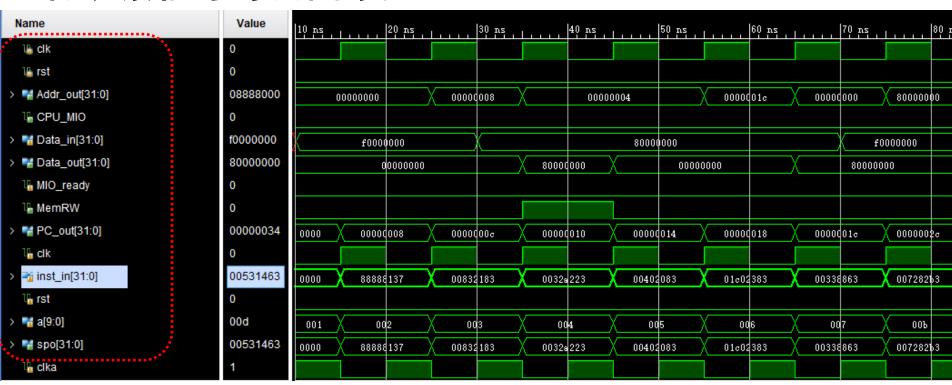
initial begin

□ 在Scope窗口下点击子模块,右键Add to wave window添加信号到波形窗口





- □观察PC地址下对应的指令内容,分析指令的执行情况判断CPU功能正确与否
- □ 待观察信号可自由添加



■ 任务二·: 设计控制器测试方案并完成测试

□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

### 物理验证

#### □使用DEMO程序目测控制器功能

- 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)
  - □ SW[10], 从0到1 (手动单步)

#### □用汇编语言设计测试程序

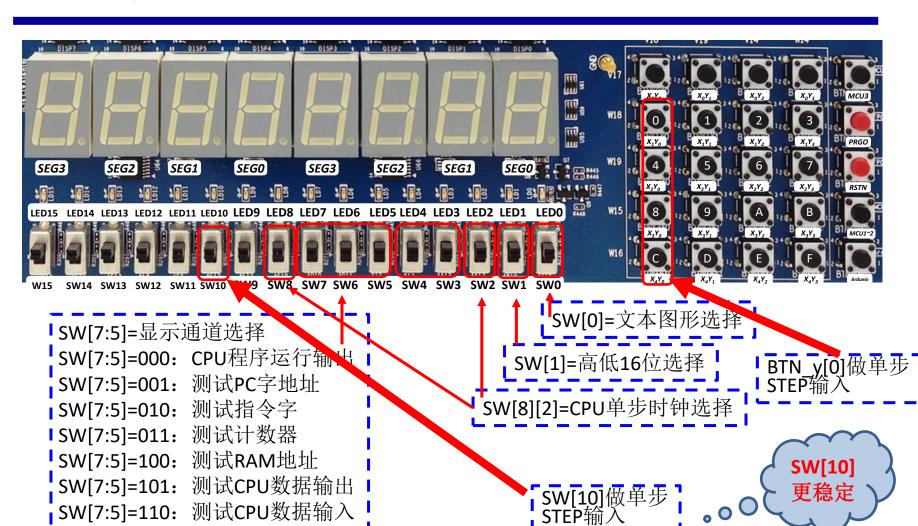
- 测试ALU指令(R-格式译码、 Function译码)
- 测试LW指令(I-格式译码)
- 测试SW指令(S-格式译码)
- 测试分支指令(B-格式译码)
- 测试转移指令(J-格式译码)

#### 物理验证

- □使用DEMO程序目测数据通路功能
  - DEMO接口功能
    - □ SW[8]=1(手动单步运行)液晶屏显示CPU信息

```
RV32I Single Cycle CPU
pc: 00000000 inst: 00100093
                                            qp: 00000000
                             sp: 00000000
             ra: 00000000
x0: 00000000
                             t2: 00000000
                                            s0: 00000000
to: 00000000
             t.1: 00000000
                                            a3: 00000000
             a1: 00000000
                             a2: 00000000
a0: 00000000
a5: 000000000 a6: 00000000
                                            s2: 00000000
                             a7: 00000000
                                            s7: 00000000
s4: 00000000 s5: 00000000
                             s6: 00000000
s9: 00000000 s10: 00000000 s11: 00000000
                                            t3: 00000000
              t6: 00000000
t5: 00000000
rs1: 00 rs1 val: 00000000
rsZ: 00 rsZ_val: 00000000
        reg_i_data: 00000000
rd: 00
                                reg_wen: 0
is imm: 0
            is_auipc: 0 is_lui: 0
                                     imm: 00000000
a val: 00000000 b val: 00000000
                                  alu_ctrl: 0
                                                cmp_ctrl:
alu_res: 00000001
                   cmp_res: 0
is_branch: 0
             is_jal: 0
                          is jalr: 0
do_branch: 0
              pc branch: 00000000
mem_wen: 0
            mem_ren: 0
dmem_o_data: f0000000
                       dmem_i_data: 00000000
                                               dmem addr:
csr_wen: 0
            csr ind: 000
                           csr_ctrl: 0
                                         csr_r_data: 00000
mstatus: 00000000 mcause: 00000000
                                    mepc: 00000000
                                                    mtual:
         00000000
mtvec:
                  mie:
                          00000000
                                    mip:
                                          00000000
```

## 物理验证-DEMO接口功能



### 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:
         addi x1,x0,1;
                           //x1=00000001
        slt x2,x0,x1;
add x3,x2,x2;
                           //x2=00000001
                           //x3=00000002
         add x4,x3,x3;
                           //x4=00000004
                           //x5=00000005
         add x5,x4,x2;
         add x6,x5,x5;
                           //x6=0000000A
         sub x7,x6,x5;
                           //x7=00000005
                           //x9=0000000A
         and x8,x7,x5;
         sub x10,x8,x6;
                           //x10=00000000
                            //x11=0000000F
         or x11,x5,x6;
         or x12,x11,x7;
                         //x12=0000000A
         slt x13,x7,x7;
                       //x13=00000000
         beg x0,x0, loop;
```

#### 测试程序参考: LW/SW

- □ 设计LW/SW程序替换DEMO程序
  - 参考04-1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: 13+x0

```
#baseAddr 0000
                           //通道结果由后一条指令读操作数观察
start:
                           //取测试常数5555555。存储器读通道
      lw x5, 0x34(x0);
start A:
                           //r1: 寄存器写通道。R5:寄存器读通道A输出
      add x1, x5, x0;
                           //r1: 寄存器读通道B输出。R2:ALU输出通道
      xor x2, x0, x1;
      lw x5, 0x48(x0); //取测试常数AAAAAAA。立即数通道:00000048
      beq x2, x5 test_sw;
                           //循环测试
                           //循环测试。
      jal x0, start;
                           //增加写SW测试,如34和48单元交换
test sw: .....
                           //循环测试。立即数通道: 0000014
      beq x0,x0,start;
```

- □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

2022/3/2 Chapter 6

48

## 动态LW/SW测试 (reserve)

#### □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E000000/FFFFFFE0
- LED显示地址是F000000/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Exp0

```
F0000000, 000002AB, 80000000, 0000003F,
                                        00000001, FFF70000,
0000FFFF,
         80000000, 00000000, 111111111, 22222222, 333333333,
4444444, 55555555, 66666666, 77777777,
                                        88888888, 99999999,
aaaaaaaa, bbbbbbbb, ccccccc, dddddddd,
                                        eeeeeee, FFFFFFF,
                                                  F7F3DFFF,
557EF7E0, D7BDFBD9, D7DBFDB9, DFCFFCFB,
                                        DFCFBFFF,
FFFFDF3D,
         FFFF9DB9, FFFFBCFB, DFCFFCFB,
                                        DFCFBFFF.
                                                  D7DB9FFF,
D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF, 03bdf020,
                                                  03def820.
08002300;
```

#### 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

- □ 单周期控制器时序体现在那里?
- □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: sra, sll,sltu;

I-Type: srai,slli,sltiu

B-Type: bne;

U-Type: lui;

- 此时用二级译码有优势吗?
- □ 动态存储模块测试七段显示会出现什么问题?

2022/3/2

