#### 第一步: 学习RISC-V指令系统



00	00000	n2 n n2 n	1 100 1d (110011 ACM 1 101 rd (110011 SML 1 100 rd (110011 SRA	
66	00000	762 N	1 116 rg 0116611 OR 1 111 rd 0116611 AND	
对这37	条整型指	令进行理解		
类型	指令名		二进制指令	作用
U	LUI	lui x1,0xffff	00001111111111111111000018110111	7820位立即設定値左移12位(低1位14年)成为一个21位数、複其 期間点。这一册令主要是方了在存存槽中乃人比较大效立即数、比 如、要包在存存地中分一个数。可则由金相令实现 d 1.d,1.00)。但这个验的范围有限(2045-2047)。因为由金相令 d 20元期除的分子14位。能表示量上的不导致2004年1月 4059、对后的有导致的范围则为2048到2047。当立即数据出这 个范围,则需要和由命令。
U	AUIPC	auipc x2,0xfff	000000001111111111111000100010111	将20位立即数的值左移12位(低12位补零)成为一个32位数,再加 上该指令的。值,再将结果写回rd。这条指令与1ui类似,可能是 为了实现心在较大范围内能转?????
J	JAL	jal x3,label1 addi x4,x0,4 label1: addi x5,x0,5	4 1 2 3	PC+4 的范斯波 rd 但不述入PC,然后计算下条册令地 址、移移地址平期限时寻址,基地地比为由指册令地 使 PC,像解形立边酸 immo 20 程序可能后的的 2 倍, 在实际区汇编时乘临中, 随林的目标往往使用汇编时中中的 label,汇编器合自动根据 label 所在的地址计算出相对的编移着赋 开格令编码。
J	JALR	addi x4,x0,4 jalr x8,x4,8 addi x5,x0,5 addi x6,x0,6 addi x7,x0,7	000000001060 <mark>00100</mark> 000 <mark>1111111100111</mark>	jair 指令使用 12 位立即数(有符号数)作为偏移量,与操作数寄存器 rsi中的值相加.然后将结果的最低有效位置0.jair指令将其下一条指令的 PC(即当前指令PC+4)的值写入其结果寄存器 rd.
В	BEQ	addi x1,x0,1 label1: add x1,x1,x1 addi x2,x0,2 beq x1,x2,label1	1111111000100000100011001 <mark>1100011</mark>	beg指令只有在操作数布存器rs1中的数值与操作数布存器rs2的数值相等时,才会跳转,跳转地址为offset的有符号扩展和最低位补0(即乘以2)的编移量加上BEQ指令的地址。
	BNE	addi x1,x0,1 label1: add x1,x1,x1 addi x2,x0,4 bne x1,x2,label1	11111110001000001001110011100011	bne指令只有在操作教育存储rs1中的教值与操作教育存储rs2的教值不相等时,才会解转,解转地址为offset的有符号扩展和最低位补O(即乘以2)的编移量加上BNE指令的地址。
	BTL	addi x1,x0,-1 label1: add x1,x1,x1 addi x2,x0,-2 blt x1,x2,label1	11111110001000001100110011100011	bt指令只有在操作数离存器rs1中的数值小于操作数离存器rs2的数值的(有符号数),才会跟转,跳转地址为offset的有符号扩展和最低位补O(即乘以2)的偏移量加上BLT指令的地址。
	BGE	addi x1, x0, -1 labell: add x1, x1, x1 addi x2, x0, -2 bge x1, x2, label	11111110001000001101110011100011	bge指令只有在場所教育存國ss1中的數值大于成場于場所教育存 國s2的數值數(特符号数),才会報報,結構地並为5m6年的時程 母子撰和繼紙位於40(即項以2)的論將運加上5GK指令的地址。
-	BLTU	1		无符号数
-	BGFII			无符号数 无符号数
_	SB			ド操作数寄存器rs2中的8位数据,写回存储器
	SH			将操作数寄存器rs2中的16位数据,写回存储器
	sw	lui x1,0xf0f11 addi		将操作数寄存器rs2中的32位数据,写回存储器
	LB	x1,x1,0x7ff sw x1,12,x0 lui x1,0x3ef12 addi x1,x1,0x7ff sw x1,12,x0 lb x2,12,x0 lb x3,13,x0 lb x4,14,x0 lb x5,15,x0		Ib哲令从存储器中读回一个8位的数据,进行符号位扩展后每回转 存器rd。
	LH	lui x1,0x3ef12 addi x1,x1,0x7ff sw x1,12,x0 lh x2,12,x0 lh x3,13,x0 lh x4,14,x0 lh x5,15,x0		IN语令从抒绪器中读回一个16位的数据,进行符号位扩展后写函数 存器rd。
	EW	lui x1,0xf0f11 addi x1,x1,0x7ff sw x1,0,x0 lw x2,0,x0		lw哲令从存储器 中读回一个32位的数据,写回寄存器rd。
_	LBU			无符号数
	LHU			无符号数
-	SLT			
-	SLTU			
-	SLIIU			
	SRL	addi x1,x0,4 addi x2,x0,-0xf0 srl x3,x2,x1		逻辑右移 (SRL) 粉型寄行器 (n2) 中的修位量対容符器 (n3) 中的値执行逻辑右移。 左边空出来的位外0. 井存梯在 (nd) 寄存器中.
	SRA			
	SLLI			
	SRLI			
	SRAI	add x2,x1,x1		add指令将寄存器 (rs1) 与寄存器 (rs2) 中的值相加,并写回 (rd) 寄存器中。
	SUB	sub x3,x2,x1		sub指令将寄存器 (rs1) 与寄存器 (rs2) 中的值相减,并写回 (rd) 寄存器中。
	ADDI	addi x1,x0,-1		addi指令将操作数离存器rs1的整数值与12位立即数进行加法操作,结果写回寄存器rd。
	XOR			

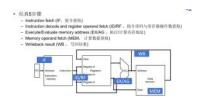
### 初步实现六条指令 JALR BEQ

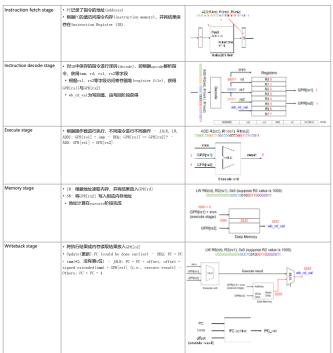
LW SW ADD

# 6条指令的功能总结

## 6条指令的功能总结

第一5年: 单周期处理器的设计 cpu包括控制器和数据通路





JALR	GPR(rs1), imm	exe_val = GPR[rs1]+ imm		pc = pc + ese_val
BEQ	GPR(ns1), GPR(ns2), mm	exe_val = GPR[rs1] == GPR[rs2]?		f (exc_val) pc = pc+mm("2)
LW	GPT(rs1), irres, rd	exe_vai = GPR(nx1)+inses	mem_val = mem_read(exe_val)	GPR[of] = merc_va
SW	GPR[nt] mm	exe_voi = GPR[n1]+imm	mem_write(exe_xal, GPR(rs2))	
ADDI	CEPT(rs1), irren, rd	exc_val = GPR(n1)+mm		GPR[vd] = ese_val
ADD	GPR[rs1], GPR[rs2], rd	asa_val = GPR(rs1(+GPR(rs2)		GPR[rd] = ess_val

储存器	类型	输入信号				输出信号		使用方法 对rom进行初始化,只需要将二进制指令存到文件"rom_binary_file_txt"中。				
指令存储 器	rom	addr (存储器读地址)			instr (读出的指令)							
数据存储器	ram	clk rst_n W_en R_en addr RW_typ e din		input input input input	时钟信号 该出的指令 与使能 该使能 该 (写) 地址 该 (写) 类型 · 字 节 · 半字 · 字 · 双字 要写入存储器的数据	dout	32bit	output	从数据存储器中读出的数据	多位宽读 其实就是		

//指令存储器	//数据存储器	'define	zero_word	32'd0
module instr_memory(	'include "define.v"		-	
addr,	module data_memory(	'define	B type	7'b1100011 //BEQ. BNE. BLT. BGE. BLTU. BGEU
instr	clk,		I type	7'b0010011 //ADDI SLTI SLTIU XORI ORI ANDI SLLI SRLI SRAI
);	rst_n,		R type	7'b0110011 //ADD SUB SLL SLT XOR SRL SRA OR AND
input [7:0]addr;	W_en,			,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
output [31:0]instr;	R_en,	'define	lui	7'b0110111
	addr,	'define	auloc	7'b0010111 //U type两条指令操作码不同
reg[31:0] rom[255:0];	RW_type, din.	Germe	umpt	7 00010111 7/0_spo-5800 4 24/1 85 1 1-5
		'define	jal	7'b1101111
//rom进行初始化	dout 1:		ialr	7b1101111 7b1100111 /// type两条指令操作码不同
initial begin	E .	define	jair	7'b1100111 //J_typeP93Rtel*/bel Farring
<pre>\$readmemb("./rom_binary_file.txt",</pre>	· · · · · · · · · · · · · · · · · · ·			
rom);	input clk; //时钟信号	'define	load	7'b0000011 //LB、LH、LW、LBU、LHU
//\$readmemh("rom_hex_file.txt",	input rst_n; //读出的指令	'define	store	7'b0100011 //SB, SH, SW
rom);				
end	input W en; //写使能	'define ADD	4'5000	1
assign instr = rom[addr]:		'define SUB	4'b001	1
assign insci – romitauuri,	input R_en; //读便能	'define SLL	4'b110	10
endmodule		'define SLT	4°b100	
enumoude	input [31:0]addr; //读写地址	'define SLTU	4'b100	
	input [2:0]RW type; //读写类型	'define XOR	4'b011	
		'define SRL	4'b110	
	input (31:0)din: //要写入存储器的数据	'define SRA	4'b111	
		'define OR	4'5010	
	output [31:0]dout; //要从数据存储器读出的数据	'define AND	4'6010	10 //这里为什么这么定义呢
				//对应的也不是ALU模块的ALU CLT?????
	reg [31:0]ram[255:0]; //数据存储器储存空间			

//定义指令代码

					wire [31:0]Rd o	data; //读的数据
					wire [31:0]W	/r data; //写的数据
					//寄存器组的工作方式	
数据通路	巻型	解羟	輸入輸出端口		'include "define.v"	ata_B; //字节拼接
					include "define.v" wire [31:0]Wr_i	data_H; //半字拼接
寄存器堆	sram 快速的	这种ram可以多路并发访问不同的寄存	信号名 位宽 输入输出 说明		clk,	
		器.	clk 1bit input 时钟信号		W_en, assign Rd_data=ram(a	addr[31:2]]; //读越准????????
	静态随	寄存器堆由32个寄存器阵列组成,其中0	Rs1 5bit input 源寄存器1		Rs1, Rs2, always⊕(*)	
	机读写	号寄存器的值恒为0,只读不写。	Rs2 5bit input 源寄存器2		Rd, begin	
	存储器		W_en 1bit input 读使能		Wr_data, case(addi	r[1:0]) //通过地址的最后两位确定数据的读法???
			Rd 5bit input 目标寄存器			00:Wr_data_B={Rd_data[31:8],din[7:0]};
			Wr_data 32bit input 写入寄存器的数据		Rd_data2 2'bi	01:Wr_data_B={Rd_data[31:16],din[7:0],Rd_data[7:0]}; 10:Wr_data_B={Rd_data[31:24],din[7:0],Rd_data[15:0]};
			Rd data1 32bit output 源寄存器1的数据			10:Wr_data_B={Rd_data[31:24],din[7:0],Rd_data[15:0]); 11:Wr_data_B={din[7:0],Rd_data[23:0]};
			Rd data2 32bit output 源寄存器2的数据		input W en; endcase	
					input [4:0]Rs1; end input [4:0]Rs2:	
译码模块		该模块对指令存储器输出的32位指令进行	信号名位宽 输入输出 说明	STREET WITH SAN	input [4:0]Rd; //半字拼接,addr[1]	路空災途所強
A-131000C		译码、得到opcode.Rs1.Rs2.Rd.imme.				ddr[1]) ? {din[15:0],Rd data[15:0]} :
		func3.func7等信息。		company to the second con-	output [31:0]Rd data1; {Rd_data[31:16],din[1	
		Tulics, Tulic 7-9 IB/0-4			output [31:0]Rd data?	
			func7 1bit output func7的第6位		//根据写类型,选择	
			Rs1 Sbit output 源寄存器1		reg [31:0] regs [31:0]; //32 12 12 13 13 13 13 13 13 13 13 13 13 13 13 13	
			Rs2 Sbit output 源奇存器2			展为32bit 数据为字节拼接 ; 等于2b'01 16bit扩展为
			Rd Sbit output 目标寄存器		//write 32bit 数据为半字拼拍	妾;
			imme 32bit output 符号扩展后的32位立即数		always@(posedge clk )	type(1:01==2'b00) ? Wr data B:((RW type(1:01==
					begin assign Wr_data=(RW 2/b01b74We-data=He	_type(1:0)==2 000) ? Wr_data_B :( (KW_type(1:0)==
ALU模块		ALU模块主要进行数据的运算,根据ALU的		(1) 加法运算还应检测是否溢	if(W_en & (RdI=0)) //武操等设施与pata目标 地址也有了 //上升沿写入数据	901765
		控制模块产生的控制信ALU_CTL决定ALU进		, 检测依据就是: 当数为负数	ADMILLENTS J	
		行的运算类型		1,最左侧的位为0,或者数为	regs[Rd]<=Wr_data; //振涛/新遊場的e_da Rd内地址对应的寄存器 begin	p=0^
		ALU_CTL 运算类型 具体的运算		数时,最左侧的位为1。	Rd内JBADS)处的条件器 <sup>200</sup> if(W_en) //	/如果写使能 wr data 写入地址为addr[9:2]的ram数据区
		0001 (我认为) 加法运算 +		体细分位以下四种情况:	//read 遊	, ANY -3 (CEE)
		0011 加法运算 -		正数+正数=负数,则溢出	assign Rd_data1=(Rs1==5'd0)?'zero_word: regs[Rs1]; raf4[830]	探路開始Wr_data;
		0100 逻辑运算 &		负数+负数=正数,则溢出	数据,如果源寄存器1地址有了那么把数据零来,如果地址没	<b>有赋</b> o
		0101 逻辑运算 或		正数-负数=负数,则溢出	assign Rd_data2=(Rs2==5'd0)?'zero_word: regs[Rs2];	
		0110 逻辑运算 ^	d.	负数-正数=正数,则溢出	endmodule //读数据	
		0111 逻辑运算 或非		(2) 逻辑运算,用逻辑门阵列	endinodule	
		1000 小于置一 无符号小于置一		现即可。		
		1001 小于置一 有符号小于置一		(3) 小于置一,实质上是减法	reg [7:0]Rd_data_B; wire [15:0]Rd_data_F	t
		1100 移位运算 逻辑左移		算。根据加法器的运算结果,		
		1101 移位运算 逻辑右移	//进	到	wire [31:0] Rd_data_i wire [31:0] Rd_data_ii	
		1110 移位运算 算数右移	ind	(d)。移位运算,这里直接使用	wife (S1.0) Rd_data_i	n_ext,
			nii	郊迎迎锋行d(sode()。	//根据地址,对数据	截取
		前两位对应运算类型			立指令 always⊕(*)	
PC寄存器		PC寄存器用以更新pc的值。	信号名位宽 输入输出 说明		提作码 begin case(addr[1:0])	
		顺序执行时,pc_new=pc+4,条件跳转时,	clk 1bit input 时钟信号	output [2:0]func3,	2'b00:Rd	_data_B=Rd_data[7:0];
		pc new=pc+imme,	rst n 1bit input 读出的指令	output func7, //fur output [4:0]Rs1,		data_B=Rd_data[15:8]; data_B=Rd_data[23:16]:
		jairfff, pc new=data(Rs1)+imme,	pc new 32bit input 下一个时钟周期的pc值	output [4:0]Rs2,		data_B=Rd_data[23:16]; data_B=Rd_data[31:24];
		January 1	pc out32bit output更新后的pc值	output [4:0]Rd,	endcase	
				output [31:0]imme ):	end	
数据通路		数据通路就是将以上的几个关键的部件进		,-	assign Rd_data_H=(ac	ddr[1])? Rd_data[31:16]:Rd_data[15:0];
		行连接,为了形成完整的数据通路,还必		wire I_type; wire U_type;		
		须添加一些 <mark>多路选择器</mark> ,作用是对不同信		wire U_type; wire J type;		82bit,RW_type[2]是用来判断是否是无符号数
		号的来源进行选择,并输出到相应的模	instr 32bit input 指令存储器读出的指令	wire 0 tune:	assign Rd_data_B_ext	t=(RW_type[2]) ? {24'd0,Rd_data_B} :

	pc_new=pc+imme。 jair時,pc_new=data(Rs1)+imme。	rst,n 1bit input 读出的概令 pc.new 32bit input 下一个创种阅播的pc值	output [4:0]Rs1, output [4:0]Rs2, output [4:0]Rs2		2'b10:Rd_data_B=Rd_data[23:16]; 2'b11:Rd_data_B=Rd_data[31:24];	
数据通路	数据通路就是将以上的几个关键的部件进		output [4:0]Rd, output [31:0]imme );		endcase end assign Rd data H=(addr[1])? Rd data[31:16]:Rd dat	:a[15:0]-
	行连接,为了形成完整的数据通路,还经 须添加一些 <mark>多路选择器</mark> ,作用是对不同信	5 clk 1 bit input 財評信号 5 rst_n 1 bit input 读出的指令	wire I_type; wire U_type; wire J_type;		//将8bit符号扩展为32bit,RW_type(2)是用来判断	是否是无符号数
	号的来源进行选择,并输出到相应的模 块。	instr 32bit input 指令存储器读出的指令 MemtoReg 1bit input 写回寄存器的数据选择器控制信号	wire B_type; wire S_type;		assign Rd_data_B_ext=(RW_type[2]) ? {24'd0,Rd_dat {{24{Rd_data_B[7]}},Rd_data_B};	ta_B}:
	别是: ALU的运算结果, 数据存储器读出	ALUSrc 1bit input ALU的数据来源的数据选择器控制信号 RegWrite 1bit input 有存器的写使能控制信号	wire [31:0]I_imme; wire [31:0]U_imme;		//16bit符号扩展为32bit, RW_type[2]是用来判断员 assign Rd_data_H_ext=(RW_type[2])? {16'd0,Rd_dat {{16[Rd_data_H[15]}},Rd_data_H};	是否是无符号数 ta_H}:
	的数据,pc+4,lui的立即数 ,auipc的 pc+imme。	lui 1bit input lui指令标志,写回寄存器的数据选择器的控制信号 U_type 1bit input U-type指令标志,写回寄存器的数据选择器的控制信号	wire [31:0]J_imme; wire [31:0]B_imme; wire [31:0]S_imme;	:	((16(KO_GATA_H(15))), KO_GATA_H); //根据读类型,选择读的数据	
	pc的来源有3个: pc+ 4,pc+imme,Read_data1+imme。	jal 1bit input ja指令标志,选择pc的数据选择器的控制信号,同时也是写回寄存器的数据选择器的控制 信号	assign opcode=instr assign func3=instr[1	(6:0); //先从32位指令中提取出7位opcode (4:12); //RISB	//等于2'b00 将8bik扩展为32bit 数据为字节拼接; 32bit 数据为半字拼接;	等于2b'01 16bit扩展为
	另外,还需要加入两个加法器,分别计算 pc+4和pc+imme。	I jair 1bit input jair指令标志,选择pc的数据选择器的控制信号,同时也是写回寄存器的数据选择器的控制信号。	assign func7=instr[3 assign Rs1=instr[19: assign Rs2=instr[24:	80]; //R :15]; //RISB	//又不是00又不是01,则为10读取或写入32bit assign dout=(RW_type[1:0]==2'b00) ? Rd_data_B_ex 2'b01) ? Rd_data_H_ext: Rd_data);	t : ((RW_type[1:0]==
		beq 1bit input beq指令标志,判断是否跳转的控制信号 bne 1bit input bne指令标志,判断是否跳转的控制信号	assign Rd =instr[11:7	7]; //RIUJ		module datapath( input clk,
		bit laik input bit於令标志,判断是否能转的控制信号 bge laik input bge指令标志,判断是否能转的控制信号 btu laik input bgth容标志,判断是否能转的控制信号	assign U_type=(instr assign J_type=(instr assign B_type=(instr	[6:0]=='jalr'   (instr[6:0]=='load)   (instr[6:0]=='l_type); r[6:0]=='lui)   (instr[6:0]=='auipc); [6:0]=='jall);	enamodule	input rst_n, input [31:0]instr,
		best little input best Minut Specific Notation (1997)  Allott Allott Allott input Allon空制信号,决定Allott Allott input Allon空制信号,决定Allott Allott input Allon空制信号,决定Allott	assign S_type=(instr	(6:0)== 'store);		input MemtoReg, input ALUSrc.
		Rd mem data 32bit input 热密存储器的地址 rom_addr 8bit output 指令存储器的地址	assign U imme={ins	{instr[31]}},instr[31:20]}; str[31:12],{12{1'b0}}};		input RegWrite, input lui, input U type,
		W_mem_data 32bit output 数据存储器的写数据 ALU_result 32bit output ALU的运算结果,作为数据存储器的读(写)地址	assign J_imme={{12} assign B_imme={{20 assign S_imme={{20	{ instr[31]}},instr[19:12],instr[20],instr[30:21],1'b0};  3[instr[31]]},instr[7],instr[30:25],instr[11:8],1'b0};  3[instr[31]]},instr[31:25],instr[11:7]);		input jal, input jalr, input beq,
		opcode 7bit output 7位操作码 func3 3bit output func3 func7 1bit output func7	assign imme= I_type	e?I_imme: //选择,指令是I_type吗? 是不是U_type?全都不是就賦()	是就用	input bne, input blt, input bge,
		Tulic / Lot Octpor fullic /	ı,	J_type?U_imme: type?I_imme: s type?B_imme:		input bltu, input bgeu, input [3:0]ALUctl,
控制器 解释			s. endinodule			input [3:0]Rd_mem_data, output [7:0]rom_addr,
器需要有技	可类型的指令所经过的数据通路不同,所以 空制信号控制数据通路,使得数据经过正确 得到正确的运算结果。	信号名位宏 輸入输出 说明 opcode 7bit input 7位操作码 func3 3bit input func3				output [31:0]Wr_mem_data, output [31:0]ALU_result, output [6:0]opcode,
本设计将	特到正确的这种后来。 特控制器分为两级控制,主控制器产生大部 制信号,子控制器是AII(控制器、产生控制	Memread 1bit output 数据存储器读使能 ALUop 2bit output 子控制器的控制信号				output [2:0]func3, output func7
	T正确运算的信号。	MemtoReg 1bit output 写回寄存器的数据选择器控制信号 Memwrite 1bit output 数据存储器写使能				);
		ALUSrc 1bit output ALU的数据来源的数据选择器控制信号 RegWrite 1bit output 衛存器的写使能控制信号				
		lui 1bit output lui指令标志,写回商存器的数据选择器的控制信号 U_type 1bit output U-type指令标志,写回商存器的数据选择器的控制信号				wire (4-f)[8-1]
		jal 1bit output jaIR令标志,选择pc的数据选择器的控制信号,同时也是写回寄存器的数据选择器的控制信 jalr 1bit output jaIR令标志。选择pc的数据选择器的控制信号,同时也是写回寄存器的数据选择器的控制信				wire [4:0]Rs1; wire [4:0]Rs2; wire [4:0]Rd; wire [31:0]imme;
		beq 1bit output beq幣令标志,判断是否解核的控制信号 bne 1bit output bm部令标志,判断是否解核的控制信号 bit 1bit output bit部令标志,判断各名解析的控制信号 bit 1bit output bit部令标本,判断各名解析的控制信号				wire [31:0] Wr_reg_data; wire [31:0] Rd_data1;
		bit bit output bit#2令标志,判断是合麻特的控制信号 bg bit output bg#2令标志,判断是合麻特的控制信号 bitu bit output blt#2令标志,判断是否麻特的控制信号				wire [31:0] Rd_data1; wire [31:0] Rd_data2; wire zero;
		But Line College (1965)				wire [31:0]pc_order;
	器根据主控制器产生的ALUop信号,结合 func7信号来产生ALUct信号。	信号名 位實 输入输出 说明				wire [31:0]pc_jump; wire [31:0]pc_new; wire [31:0]pc_out;
ALUop ∃	子控制器的操作 加法运算	ALUOp 2bit input 主控制器产生的ALUOp信号 func3 3bit input func3				wire jump_flag;
10 R	R-type指令,根据func3和func7判断运算类型 R-type指令,根据func3和func7判断运算类型	func7 7bit input func7  ALUCH 4bit output 子控制器产生的ALUop语号				wire [31:0]ALU_DB; wire [31:0]WB_data;
11 🕯	条件跳转信号,根据func3判断运算类型 1号设置如下:					wire reg_sel; wire [31:0]Wr_reg_data1; wire [31:0]Wr_reg_data2;
ALUcti ji	-	invr[25] [15   15   15   16   11   11   14   11   11   12   12   12				wire [31:0]pc_jump_order; wire [31:0]pc_jalr;
0011 D	加法运算(真的不是0001???) 加法运算	mm(21)05   m2   m1   000   mm(41)11   10001   BM:   mm(21)05   m2   m1   100   mm(41)11   100011   BLT     mm(21)05   m2   n1   101   mm(41)11   110001   BLT				assign reg_sel=jal   jalr ;
0100 N 0101 N 0110 N	逻辑运算					assign Wr_mem_data=Rd_data2; assign rom_addr=pc_out[9:2]; assign pc_jalr={ALU_result[31:1],1'b0};
0110 10 0111 13 1000 /1	逻辑运算	1				pc_reg pc_reg_inst ( .clk(clk),
1000 1	小于置一	ment 1.5  552 c51 666 cman 4.00 0160611 S41   ment 1.5  552 c51 000 cman 4.00 0160611 S40   ment 2.100 c51 000 cft 0010611 SW   ment 2.100 c51 000 cft 0010611 ADDI   ment 2.100 cft 0010611 St.T1				.rst_n(rst_n), .pc_new(pc_new), .pc_out(pc_out)
1101 R	移位运算	### (12.00   15.00   1				);
						instr_decode instr_decode_inst ( .instr(instr), .opcode(opcode),
		0000000   n2   n1   000   rt   011001   St.B.				.func3(func3), .func7(func7), .Rs1(Rs1),
						.Rs2(Rs2), .Rd(Rd), .imme(imme)
	制模块和子控制模块进行实例化,得到控制	0000000 D2 O1 III # 0110011 AND				registers registers_inst ( .clk(clk).
模块。	NOWANTIINONALIIANII, MAUIN					.W_en(RegWrite), .Rs1(Rs1), .Rs2(Rs2),
ok,上面步骤完E	成之后,会拥有以下文件					.Rd(Rd), .Wr_data(Wr_reg_data), .Rd_data1(Rd_data1),
aluv al control v	2022/11/6 20:04 2022/11/6 20:51	V 2/時 V 2/時				.Rd_data2(Rd_data2) );
al data memory: al datapath.v al define.v	xw 2022/11/614:01 2022/11/620:17 2022/11/614:36	<ul><li>ソ 次件</li><li>ソ 次件</li><li>ソ 次性</li></ul>				alu alu_inst ( _ALU_DA(Rd_data1),
al instr_decode.al instr_memory:	v 2022/11/61448 tw 2022/11/613:03	V 文件 V 文件				.ALU_DB(ALU_DB), .ALU_CTL(ALUctl), .ALU_ZERO(zero),
al pc_reg.v al pc_reg.v al registers.v	2022/11/6 20:06 2022/11/6 16:17 2022/11/6 14:17	Y 20年 Y 20日 Y 20日				_ALU_OverFlow(), _ALU_DC(ALU_result) );
图 rom_binary_file 指令寄存器,指令	le.txt 2022/11/6:12:47	文本文档				branch_judge_branch_judge_inst (
数据存储器,数据 计算单元						.bne(bne), .bit(bit), .bge(bge),
数据通路,多路边 pc寄存器						.bitu(bitu), .bgeu(bgeu), .jal(jal),
控制器, 计算控制 但是他们现在是分	制器 分散的模块,需要将他们连接起来					.jalr(jalr), .zero(zero), .ALU_result(ALU_result),
	OF EAST STATE OF THE STATE OF T					.jump_flag(jump_flag) );
RISC-V核心 信号名 位宽 clk 1bit	- 将数据通路和控制信号连接 输入输出 说明 input 系统时钟					
rst_n 1bit instr 32bit	input 复位信号				,	//pc+4 cla_adder32 pc_adder_4 ( _A(pc_out),
Rd_mem_data 32	2bit input 来自数据存储器的读数据 bit output 指令存储器的读地址					.B(32'd4), .cin(1'd0), .result(pc_order),
	2bit output 写入数据存储器的数据 output 数据存储器的读使能信号					.cout() );
	2bit output 数据存储器的读 (写) 地址				,	/pc+imme cla_adder32 pc_adder_imme ( _A(pc_out),
RW_type 3b	bit output 数据存储器的读写类型					.B(imme), .cin(1'd0), .result(pc_jump),
aluv	2022/11/6 20:04	V.29h				.cout() );
a control v a data_memory.v a datapath.v	2022/11/6 20:51 2022/11/6 14:01 2022/11/6 20:17	V 25F V 25R V 25R			,	/pc_sel mux pc_mux (
define.v	2022/11/6 14:36 v 2022/11/6 14:48	V 284 V 284				.data1(pc_jump), .data2(pc_order), .sel(jump_flag), .dautic_jump_order)
instr_memory.v mus.v ipc_reg.v	2022/11/6 12:03 2022/11/6 20:06 2022/11/6 16:17	V 22H V 22H V 25H			,	.dout(pc_jump_order) ); ///pc_jair
ineglisters.v	2022/11/6 14:17 2022/11/6 21:02	V 294 V 284				mux pc_jair_mux ( .data1(pc_jair), .data2(pc_jump_order),
inscv_top.v	2022/11/6 21:03 le.tet 2022/11/6 12:47	V 交換 文本交換				.sel(jalr), .dout(pc_new) );
	之后相当于,将控制器、数据选择器、数据右 u的理论设计,下面开始尝试仿真	<b>储器、指令寄存器结合了起来</b>				Allidata sel
" Error: (vlog-	p-13069) Dr\modeleim\eLngle_oyole_opu\als.	V(284): oper "1": #355ax			,	ALUdata_sel mux ALU_data_mux ( .data1(imme), .data2(Rd_data2),
error, unexpect	ted ' '.					.data2(Rd_data2), .sel(ALUSrc), .dout(ALU_DB) );
282 //后面看 283 //assig 284 //	m TATA TO THE TOTAL TOT	200011(大)4(U, CT. 指子[25] 4 ACD_resolt[13]) //正数亚亚坎·贝斯,则应出 4 A[3] 5 4 [3] 8 4 CO_resolt[3]) //元数亚亚坎·贝斯 8 A[3] 5 4 [3] 8 4 CO_resolt[32]) //克数·亚敦·亚敦 8 A[3] 5 4 [3] 8 4 CO_resolt[32]) // 克敦·亚敦·亚敦				/ALU result or datamem
285 // 286 //	((ALU_CTL==d'b0011)  ((ALU_CTL==d'b0011) 8	ā A[31] ā -□[31] ā -□CO_result[31]) // 负数-正数-正数 -a[31] ā d[s1] ā MOO result[31]); // 正数-负数-数数, 形図出			,	mux WB_data_mux ( data1(Rd mem data)

```
** Error: (vlog-1306) Ditmodelsim/single_oyole_oputals.v(384): near *(*) syntax error, unaspected *(*).
                                                                                                   odelsim/single_cycle_cpu\als.v(254): near "endmodule"
endmodule, expecting IDENTIFIER or TYPE_IDENTIFIER or
                                    从libary进入simulation
      vaim -qui

5 Deart time: 21:55:09 on Nov 06,2022

5 ** Note: (vaim-9812) Design is being optimized...
    意料之内的func7的宽度问题,是我智障了
Edit, 作子総修正常の頂了

The Table View Add Formed Tools Sockenpiks Window Help

The Life View Add Formed Tools Sockenpiks Window Help

The Life View Add Formed Tools Sockenpiks Window Help

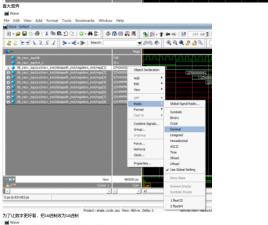
The Company of t
   正常仿真这话说早了,在这之后,又花了1h去研究这个testbench要写到哪里,网上能直到的只有
                                  'timescale 1ns / 1ps
                                      module tb_riscv_top;
                                                     // Inputs
reg clk;
reg rst_n;
                                                   // Outputs
wire [7:0] rom_addr;
                                                                // Add stimulus here
                                                end
    但是,我没整明白,要执行的指令放到哪里呢?
```

data Jimme),
data Jimme),
data Jimed data Jimed
data Jimed data Jimed
data Jimed data Jimed
max Will, data Jimed
data Jimed data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jimed
data Jim

# 最后,突然想到之前的二进制文件,也就是rom的来源 于是把指令翻译成二进制文件放进去

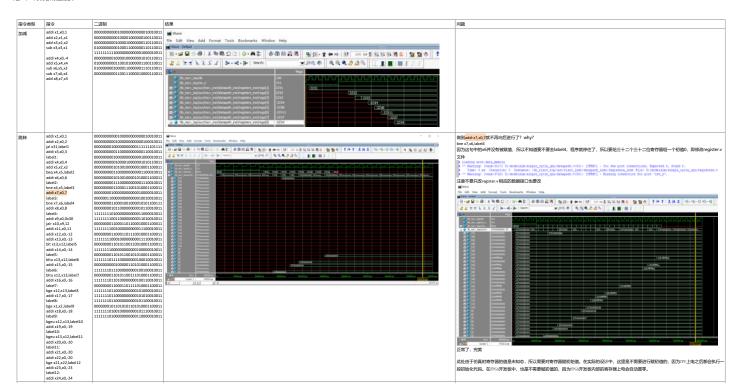
E00E00E00100E10E00E00110110E11 0100E00E000000100E1000E00110110C11 1111111111000E000E000E0100E00E01E01 E00E000E01E00E10E00EE00E0110F110E11 010E000E001E00E10E00E00E0110F110C11 E10E00E001E00E11E00E00110E110E11 E00E00E00E00110E11100E010E00110E11

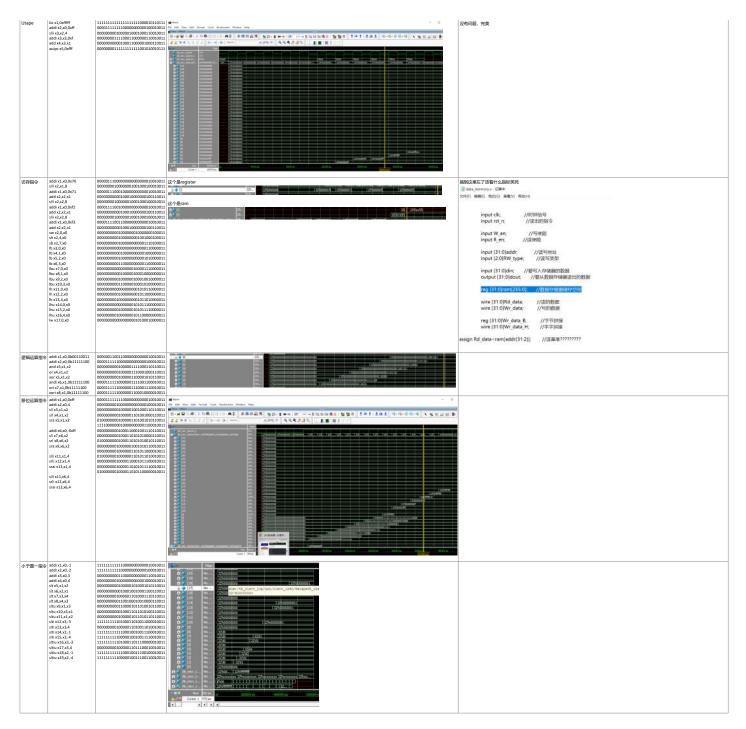
## 成功了 Wave



関Visco His Cate Visco Add former Tools Sockmarks Window into Cate Visco Add former Tools Sockmarks Window into Cate Add Add former Tools Sockmarks Window into Cate Add Into Cate Add

可见,算数指令执行的没有问题 下面一个一个执行各种类型的指令





好消息好消息,至此单周期cpu算是大功告成,下面终于进入正题,流水线设计