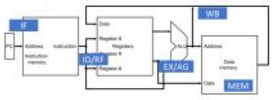
多周期流水线

2022年11月7日 12:37

cpu包括控制器和数据通路

- 经典5步骤
  - Instruction fetch (IF，指令获取)
  - Instruction decode and register operand fetch (ID/RF，指令译码与寄存器操作数获取)
  - Execute/Evaluate memory address (EX/AG，执行/计算内存地址)
  - Memory operand fetch (MEM，计算数据获取)
  - Writeback result (WB，写回结果)



Instruction fetch stage	• PC记录了指令的地址(address) • 根据PC的值访问指令内存(instruction memory)，并将结果保存在Instruction Register (IR)。
Instruction decode stage	• 对IR中保存的指令进行译码(decode)，即根据opcode解析指令，获得imm，rd，rs1，rs2等字段 • 根据rs1，rs2等字段访问寄存器堆(register file)，获得GPR[rs1]与GPR[rs2] • wb_rd_val为写回值，由写回阶段获得
Execute stage	• 根据操作数进行执行，不同指令进行不同操作 - JALR, LW, ADDI: GPR[rs1] + imm - BEQ: GPR[rs1] == GPR[rs2]? - ADD: GPR[rs1] + GPR[rs2]
Memory stage	• LW：根据地址读取内存，并将结果放入GPR[rd] • SW：将GPR[rs2] 写入相应内存地址 • 地址计算在execute阶段完成
Writeback stage	• 将执行结果或内存读取结果放入GPR[rd] • Update(更新) PC (could be done earlier) - BEQ: PC = PC + imm(*2，没有第0位) - JALR: PC = PC + offset, offset = signed extended(imm) + GPR[rs1] (i.e., execute result) - Others: PC = PC + 4

	Decode	Execute	Memory	Write back
JALR	GPR[rs1], imm	exe_val = GPR[rs1]+ imm		pc = pc + exe_val
BEQ	GPR[rs1], GPR[rs2], imm	exe_val = GPR[rs1] == GPR[rs2]?		if (exe_val) pc = pc+imm(*2)
LW	GPR[rs1], imm, rd	exe_val = GPR[rs1]+imm	mem_val = mem_read(exe_val)	GPR[rd] = mem_val
SW	GPR[rs1], GPR[rs2], imm	exe_val = GPR[rs1]+imm	mem_write(exe_val, GPR[rs2])	
ADDI	GPR[rs1], imm, rd	exe_val = GPR[rs1]+imm		GPR[rd] = exe_val
ADD	GPR[rs1], GPR[rs2], rd	exe_val = GPR[rs1]+GPR[rs2]		GPR[rd] = exe_val

数据通路

在两个子模块之间添加一组流水线寄存器。

其作用有二：

（1）用寄存器打断组合逻辑

（2）进行信号在两个子模块之间的传递

其信号传递关系如下：

IF	IF/ID	ID	ID/EX	EX	EX/MEM	MEM	MEM/WB	WB
pc	pc		pc					
instr	instr							
		imme	imme	imme	imme		imme	
		Rd_data1	Rd_data1					
		Rd_data2	Rd_data2	Rd_data2	Rd_data2			
		Rd	Rd		Rd		Rd	
		func3						
		func7						
		opcode						
				ALU_result	ALU_result		ALU_result	
				pc_jump	pc_jump		pc_jump	
				pc_order	pc_order		pc_order	
				pc_new				
						loaddata	loaddata	
							Wr_reg_data	

阶段	功能	涉及的文件	
IF	产生新的pc值，并读取指令存储器。因此取指阶段包括pc_reg和instr_memory两个子模块。此处 instr_memory为一个存储器，将其当作FPGA外部的部件而不将其写入if_stage阶段。	if_stage.v 2022/11/7 12:47 V 文件 instr_memory.v 2022/11/6 13:03 V 文件 pc_reg.v 2022/11/7 12:46 V 文件	module if_stage(input clk, input rst_n, input [31:0]pc_if_i, output [31:0]pc_if_o, output [7:0]inst_addr); endmodule
IF_ID流水线寄存器	此模块传递两个信号，分别是pc和读出的32位的指令。 pc为什么向后传递？因为在执行阶段计算下一个pc的值以及跳转指令的目标地址时，都需要用到当前模块的pc,所以pc需要逐级传递，保证pc与所在阶段执行的指令是一一对应的。	if_id_regs.v 2022/11/7 12:51 V 文件	module if_id_regs(input clk, input rst_n, input [31:0]pc_if_id_i, input [31:0]instr_if_id_i, output reg [31:0]pc_if_id_o, output reg [31:0]instr_if_id_o); endmodule
ID	译码阶段的功能是将上一级传递的32位指令进行译码并且读取寄存器堆。所以此模块包含instr_decode和registers两个子模块	id_stage.v 2022/11/7 12:54 V 文件 if_id_regs.v 2022/11/7 12:51 V 文件 if_stage.v 2022/11/7 12:47 V 文件 instr_decode.v 2022/11/6 23:43 V 文件 instr_memory.v 2022/11/6 13:03 V 文件 pc_reg.v 2022/11/7 12:46 V 文件 registers.v 2022/11/7 11:44 V 文件	module id_stage(input clk, input rst_n, input RegWrite_id_i, input [31:0]Wr_reg_data_id_i, input [31:0] instr_id_i, output [6:0] opcode_id_o, output [2:0] func3_id_o, output func7_id_o, output [31:0] imme_id_o, output [31:0] Rd_data1_id_o, output [31:0] Rd_data2_id_o); wire [4:0]Rs1; wire [4:0]Rs2; wire [4:0]Rd;

ID_EX流水线寄存器	此模块传递四个信号，pc,imme,Rd_data1,Rd_data2。	id_ex_regs.v 2022/11/7 12:59 V 文件	`include "define.v" module id_ex_regs(input clk, input rst_n, input [31:0]pc_id_ex_i, input [31:0]imme_id_ex_i, input [31:0]Rd_data1_id_ex_i, input [31:0]Rd_data2_id_ex_i, output reg [31:0]pc_id_ex_o, output reg [31:0]imme_id_ex_o, output reg [31:0]Rd_data1_id_ex_o, output reg [31:0]Rd_data2_id_ex_o,);
EX	执行阶段的功能是执行ALU的计算并且计算新的pc的值。 所以包含alu模块，分支判断模块，两个加法器（pc+4,pc+imme），三个选择器（alu数据来源选择，pc顺序执行与pc跳转选择，jalr选择）。	alu.v 2022/11/6 21:26 V 文件 branch_judge.v 2022/11/7 0:11 V 文件 ex_stage.v 2022/11/7 13:00 V 文件 id_ex_regs.v 2022/11/7 12:59 V 文件 id_stage.v 2022/11/7 12:54 V 文件 if_id_regs.v 2022/11/7 12:51 V 文件 if_stage.v 2022/11/7 12:47 V 文件 instr_decode.v 2022/11/6 23:43 V 文件 instr_memory.v 2022/11/6 13:03 V 文件 mux.v 2022/11/6 20:06 V 文件 pc_reg.v 2022/11/7 12:46 V 文件 registers.v 2022/11/7 11:44 V 文件	module ex_stage(input ALUctl_ex_i, input beq_ex_i, input bne_ex_i, input blt_ex_i, input bge_ex_i, input bltu_ex_i, input bgeu_ex_i, input jal_ex_i, input jalr_ex_i, input ALUSrc_ex_i, input [31:0]pc_ex_i, input [31:0]imme_ex_i, input [31:0]Rd_data1_ex_i, input [31:0]Rd_data2_ex_i, output [31:0]ALU_result_ex_o, output [31:0]pc_new_ex_o, output [31:0]pc_jump_o, output [31:0]Rd_data2_ex_o, output [31:0]imme_ex_o, output [31:0]pc_order_ex_o); wire [31:0]ALU_DB; wire zero; wire ALU_result_sig; wire jump_flag; wire [31:0]pc_order; wire [31:0]pc_jump_order; wire pc_jalr;
EX_MEM流水线寄存器	此模块传递ALU_result，pc_jump，Rd_data2，imme，pc_order五个模块。 ALU_result将在访存阶段作为访存的地址使用，在写回阶段作为运算结果写回寄存器。 pc_jump，pc_order在写回阶段写回寄存器。 Rd_data2在访存阶段作为写入存储器的数据使用。 imme在写回阶段使用。	ex_mem_regs.v 2022/11/7 13:04 V 文件	`include "define.v" module ex_mem_regs(input clk, input rst_n, input [31:0]ALU_result_ex_mem_i, input [31:0]pc_jump_ex_mem_i, input [31:0]Rd_data2_ex_mem_i, input [31:0]imme_ex_mem_i, input [31:0]pc_order_ex_mem_i, output [31:0]ALU_result_ex_mem_o, output [31:0]pc_jump_ex_mem_o, output [31:0]Rd_data2_ex_mem_o, //DM output [31:0]imme_ex_mem_o, output [31:0]pc_order_ex_mem_o,);
MEM	访存阶段与取指阶段类似，都是访问外部存储器，在这里将存储器与CPU进行分开，所以此阶段只需要输出地址与数据给数据存储器，并且接收数据存储器读出的数据，此部分代码在CPU的顶层模块体现。		
MEM_WB流水线寄存器	此模块传递5个信号，均在写回阶段使用。loaddata为访存阶段新产生的数据，来自于读数据存储器。	mem_wb_regs.v 2022/11/7 13:07 V 文件	module mem_wb_regs(input [31:0]ALU_result_mem_wb_i, input [31:0]pc_jump_mem_wb_i, input [31:0]loaddata_mem_wb_i, //DM input [31:0]imme_mem_wb_i, input [31:0]pc_order_mem_wb_i, output reg [31:0]ALU_result_mem_wb_o, output reg [31:0]pc_jump_mem_wb_o, output reg [31:0]loaddata_mem_wb_o, //DM output reg [31:0]imme_mem_wb_o, output reg [31:0]pc_order_mem_wb_o,);
WB	写回阶段包括4个选择器，最终输出一个数据写回寄存器堆。	mux.v 2022/11/6 20:06 V 文件 pc_reg.v 2022/11/7 12:46 V 文件 registers.v 2022/11/7 11:44 V 文件 wb_stage.v 2022/11/7 13:10 V 文件	module wb_stage(input MemtoReg, input jal, input jalr, input lui, input U_type, input [31:0]ALU_result_wb_i, input [31:0]pc_jump_wb_i, input [31:0]loaddata_wb_i, input [31:0]imme_wb_i, input [31:0]pc_order_wb_i, output [31:0]Wr_reg_data_wb_o); wire [31:0]WB_data; wire reg_sel; wire [31:0]Wr_reg_data1; wire [31:0]Wr_reg_data2;

数据通路：将以上五个子模块以及四个流水线寄存器组进行实例化，将模块与模块之间的信号进行连接，即得到数据通路部分。

datapath.v 2022/11/7 13:12 V 文件

控制器

IF	IF/ID	ID	ID/EX	EX	EX/MEM	MEM	MEM/WB	WB
pc	pc		pc					
instr	instr							
		imme	imme	imme	imme		imme	
		Rd_data1	Rd_data1					
		Rd_data2	Rd_data2	Rd_data2	Rd_data2			
		Rd	Rd		Rd		Rd	
		func3						
		func7						
		opcode						
				ALU_result	ALU_result		ALU_result	
				pc_jump	pc_jump		pc_jump	
				pc_order	pc_order		pc_order	
				pc_new				
						loaddata	loaddata	
								Wr_reg_data
		ALUSrc	ALUSrc					
		ALUctl	ALUctl					
		beq	beq					
		bne	bne					
		blt	blt					
		bge	bge					
		bltu	bltu					
		bgeu	bgeu					
		jal	jal	jal	jal	jal	jal	
		jalr	jalr	jalr	jalr	jalr	jalr	
		MemRead	MemRead	MemRead	MemRead			
		MemWrite	MemWrite	MemWrite	MemWrite			
		RW_type	RW_type	RW_type	RW_type			
		lui	lui	lui	lui	lui	lui	
		U_type	U_type	U_type	U_type	U_type	U_type	
		MemtoReg	MemtoReg	MemtoReg	MemtoReg	MemtoReg	MemtoReg	
		RegWrite	RegWrite	RegWrite	RegWrite	RegWrite	RegWrite	RegWrite

首先在译码阶段产生完整的一组控制信号（control模块的输出）
然后将这一组控制信号输入到ID/EX流水线寄存器组进行延迟一拍。
在执行阶段，需要用到ALUsrc,ALUctl,beq,bne,blt,bge,bltu,bgeu,jal,jalr信号。
但是由于jal和jalr信号在写回阶段需要再次用到，所以将jal和jalr信号随同剩下的信号一起输入到EX/MEM流水线寄存器并延迟一拍。
在访存阶段，要用掉三个信号，MemRead，MemWrite，RW_type,
并将剩下的信号输入到MEM/WB流水线寄存器。
在写回阶段，lui,U_type,MemtoReg用作选择器的选择端，RegWrite输入到寄存器堆。
以上就是控制信号的传递，实际上只需要按照上一篇文章的思路扩展流水线寄存器即可

ID_EX流水线寄存器	//control signals input ALUSrc_id_ex_i, input [3:0]ALUctl_id_ex_i, input beq_id_ex_i, input bne_id_ex_i, input blt_id_ex_i, input bge_id_ex_i, input bltu_id_ex_i, input bgeu_id_ex_i, input jal_id_ex_i, input jalr_id_ex_i, input MemRead_id_ex_i, input MemWrite_id_ex_i, input [2:0]RW_type_id_ex_i, input lui_id_ex_i, input U_type_id_ex_i, input MemtoReg_id_ex_i, input RegWrite_id_ex_i, output reg ALUSrc_id_ex_o, output reg [3:0]ALUctl_id_ex_o, output reg beq_id_ex_o, output reg bne_id_ex_o, output reg blt_id_ex_o, output reg bge_id_ex_o, output reg bltu_id_ex_o, output reg bgeu_id_ex_o, output reg jal_id_ex_o, output reg jalr_id_ex_o, output reg MemRead_id_ex_o, output reg MemWrite_id_ex_o, output reg [2:0]RW_type_id_ex_o, output reg lui_id_ex_o, output reg U_type_id_ex_o, output reg MemtoReg_id_ex_o, output reg RegWrite_id_ex_o		
EX_MEM流水线寄存器	//control signals input jal_ex_mem_i, input jalr_ex_mem_i, input MemRead_ex_mem_i, input MemWrite_ex_mem_i, input [2:0]RW_type_ex_mem_i, input lui_ex_mem_i, input U_type_ex_mem_i, input MemtoReg_ex_mem_i, input RegWrite_ex_mem_i, output reg jal_ex_mem_o, output reg jalr_ex_mem_o, output reg MemRead_ex_mem_o, output reg MemWrite_ex_mem_o, output reg [2:0] RW_type_ex_mem_o, output reg lui_ex_mem_o, output reg U_type_ex_mem_o, output reg MemtoReg_ex_mem_o, output reg RegWrite_ex_mem_o		
MEM_WB流水线寄存器	//control signals input jal_mem_wb_i, input jalr_mem_wb_i, input lui_mem_wb_i, input U_type_mem_wb_i, input MemtoReg_mem_wb_i, input RegWrite_mem_wb_i, output reg jal_mem_wb_o, output reg jalr_mem_wb_o, output reg lui_mem_wb_o, output reg U_type_mem_wb_o, output reg MemtoReg_mem_wb_o, output reg RegWrite_mem_wb_o		

顶层模块的设计

1.将流水线寄存器替换（例化接口啥的改变了，加入了控制信号）

2.将各个模块的控制信号接入

3.留意几个特殊的信号：

(1).访存阶段的控制信号R_en,W_en,RW_type要输出到CPU外部的数据存储器,之前直接从control模块输出，现在需要将其延迟两拍，从EX/MEM流水线寄存器输出，也就是说，在顶层模块中这组信号是从数据通路中输出来的而不是从控制器模块输出来的，（这里不是很懂）

(2).译码阶段的寄存器堆模块的输入信号，Wr_reg_data和RegWrite，均来自写回阶段。

绝了报warning报出阴影了

ok了改改后面的接口和接口长度的设置错误之后终于开始仿真了

```
VSIM 33> quit -sim
# reading D:/software/modelsim/win64/../modelsim.ini
# loading project pipeline
# Load canceled
# Compile of alu.v was successful.
# Compile of branch_judge.v was successful.
# Compile of datapath.v failed with 4 errors.
# Compile of ex_mem_regs.v failed with 27 errors.
# Compile of ex_stage.v failed with 2 errors.
# Compile of id_stage.v was successful.
# Compile of if_id_regs.v failed with 5 errors.
# Compile of if_stage.v was successful.
# Compile of instr_decode.v failed with 14 errors.
# Compile of instr_memory.v was successful.
# Compile of mem_wb_regs.v failed with 23 errors.
# Compile of mux.v was successful.
# Compile of pc_reg.v failed with 3 errors.
# Compile of registers.v failed with 49 errors.
# Compile of riscv.v was successful.
# Compile of riscv_top.v was successful.
# Compile of tb_riscv_top.v was successful.
# Compile of wb_stage.v failed with 2 errors.
# 18 compiles, 9 failed with 149 errors.
```

啊啊啊啊啊啊全是error哪不住了

```
vlog -work work -vopt -stats=none D:/modelsim/pipeline/ex_stage.v
Model Technology ModelSim SE-64 vlog 10.4 Compiler 2014.12 Dec  3 2014
-- Compiling module ex_stage
** Error: D:/modelsim/pipeline/ex_stage.v(33): (vlog-2730) Undefined variable: 'ALU_result'.

** Error: (vlog-13069) D:/modelsim/pipeline/ex_stage.v(69): near ",": syntax error, unexpected ',', expecting ')'.
```

```
input [31:0]Rd_data1_ex_i,
input [31:0]Rd_data2_ex_i,
output [31:0]ALU_result_ex_o,
output [31:0]pc_new_ex_o,
output [31:0]pc_jump_o,
output [31:0]Rd_data2_ex_o,
output [31:0]imme_ex_o,
output [31:0]pc_order_ex_o
);

wire [31:0]ALU_DB;
wire zero;
wire ALU_result_sig;
wire jump_flag;
wire [31:0]pc_order;
wire [31:0]pc_jump_order;
wire pc_jalr;


assign pc_jalr={ALU_result[31:1],1'b0};
assign ALU_result_sig=ALU_result[31];
assign imme_ex_o=imme_ex_i;
assign pc_order_ex_o=pc_order;

alu alu_inst (
.ALU_DA(Rd_data1_ex_i),
.ALU_DB(ALU_DB),
.ALU_CTL(ALUctl_ex_i),
.ALU_ZERO(zero),
.ALU_Overflow(),
.ALU_DC(ALU_result_ex_o)
);
```

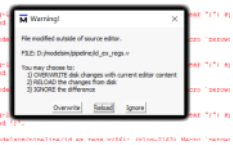
ALU输出结果改名了

```
assign pc_jalr={ALU_result_ex_o[31:1],1'b0};
assign ALU_result_sig=ALU_result_ex_o[31];
assign imme_ex_o=imme_ex_i;
assign pc_order_ex_o=pc_order;
```

```
# Compile of alu.v was successful.
# Compile of branch_judge.v was successful.
# Compile of datapath.v was successful.
# Compile of ex_mem_regs.v was successful.
# Compile of ex_stage.v was successful.
# Compile of id_stage.v was successful.
# Compile of if_id_regs.v was successful.
# Compile of if_stage.v was successful.
# Compile of instr_decode.v was successful.
# Compile of instr_memory.v was successful.
# Compile of mem_wb_regs.v was successful.
# Compile of mux.v was successful.
# Compile of pc_reg.v was successful.
# Compile of registers.v was successful.
# Compile of riscv.v was successful.
# Compile of riscv_top.v was successful.
# Compile of tb_riscv_top.v was successful.
# Compile of wb_stage.v was successful.
# Compile of define.v was successful.
# 19 compiles, 0 failed with no errors.
```

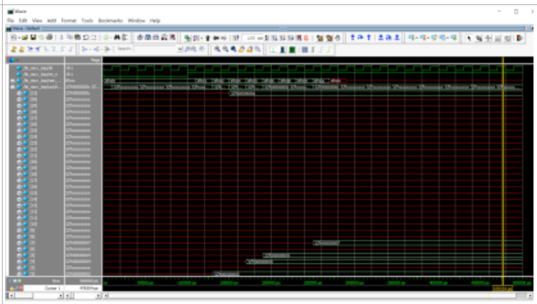
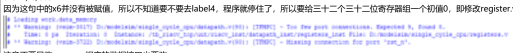
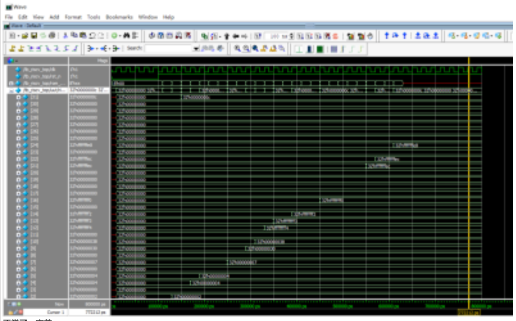
其他都是sysnax错误没意思

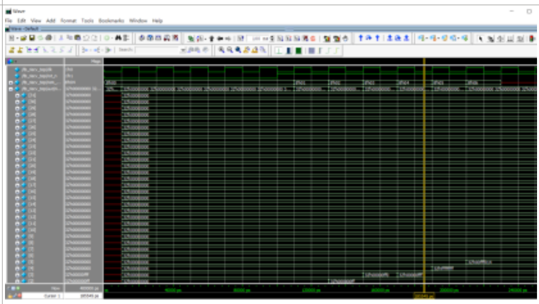
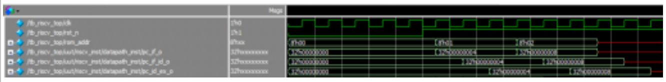
```
logy ModelSim SE-64 vlog 10.4 Compiler 2014.12 Dec  3 2014
 module id_ex_regs
/modelsim/pipeline/id_ex_regs.v(62): (vlog-2163) Macro 'zerow
```



指令类型	指令	二进制	结果	问题

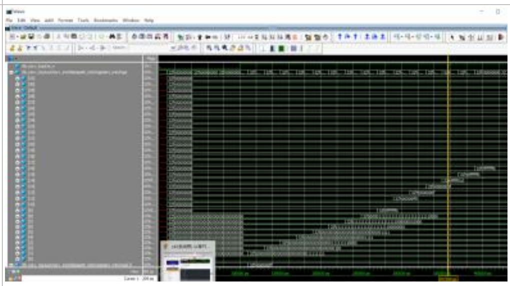
加减	addi x1,x0,1
	add x2,x1,x1
	add x3,x2,x2
	sub x3,x3,x1
	
	addi x4,x0,-4
	add x5,x4,x4
	sub x6,x5,x3
	sub x7,x6,x4
	add x8,x7,x3
		00000000000100000000000010010011
		00000000000100001000000100110011
		00000000001000010000000110110011
		01000000000100011000000110110011
		11111111110000000000001000010011
		00000000010000100000001010110011
		01000000001100101000001100110011
		01000000010000110000001110110011
		00000000001100111000010000110011
				可以看见，计算的答案错了，是数据依赖没有处理

跳转	addi x1,x0,1
	addi x2,x0,2
	jal x31,label1
	addi x3,x0,3
	label1:
	addi x4,x0,4
	add x5,x2,x2
	beq x4,x5,label2
	addi x6,x0,6
	label2:
	bne x4,x5,label3
	addi x7,x0,7
	label3:
	bne x7,x6,label4
	addi x8,x0,8
	label4:
	addi x9,x0,0x30
	jalr x10,x9,12
	addi x11,x0,11
	addi x12,x0,-12
	addi x13,x0,-13
	blt x13,x12,label5
	addi x14,x0,-14
	label5:
	bltu x13,x12,label6
	addi x15,x0,-15
	label6:
	bltu x12,x13,label7
	addi x16,x0,-16
	label7:
	bge x12,x13,label8
	addi x17,x0,-17
	label8:
	bge x1,x2,label9
	addi x18,x0,-18
	label9:
	bgeu x12,x13,label10
	addi x19,x0,-19
	label10:
	bgeu x13,x12,label11
	addi x20,x0,-20
	label11:
	addi x21,x0,-20
	addi x22,x0,-20
	bge x21,x22,label12
	addi x23,x0,-23
	label12:
	addi x24,x0,-24
		00000000000100000000000010010011
		00000000001000000000000100010011
		00000000100000000000111111101111
		00000000001100000000000110010011
		00000000010000000000001000010011
		00000000001000010000001010110011
		00000000010100100000010001100011
		00000000011000000000001100010011
		00000000010100100001010001100011
		00000000011100000000001110010011
		00000000011000111001010001100011
		00000000100000000000010000010011
		00000011000000000000010010010011
		00000000110001001000010101100111
		00000000101100000000010110010011
		11111111010000000000011000010011
		11111111001100000000011010010011
		00000000110001101100010001100011
		11111111001000000000011100010011
		00000000110001101110010001100011
		11111111000100000000011110010011
		00000000110101100101010001100011
		11111111000000000000100000010011
		00000000110101100101010001100011
		11111110111000000000100010010011
		00000000000100001101010001100011
		11111110111000000000100100010011
		00000000110001101111010001100011
		11111110110100000000100110010011
		00000000110001101111010001100011
		11111110110000000000101000010011
		11111110110000000000101010010011
		11111110110000000000101100010011
		00000001011010101101010001100011
		11111110100100000000101110010011
		11111110100000000000110000010011
				做到addi x7,x0,7就不再向后进行了？why?
				bne x7,x6,label4
				因为这句中的x6并没有被赋值，所以不知道要不要去label4，程序就停住了，所以要给三十二个三十二位寄存器组一个初值0，即修改register.v文件
				 
				
				注意不要只改register.v 相应的数据接口也要改
				 
				正常了，完美
				
				此处由于仿真时寄存器的值是未知态，所以需要对寄存器赋初始值。在实际的设计中，这里是不需要进行赋初值的，因为CPU上电之后都会执行一段初始化代码。在FPGA开发板中，也是不需要赋初值的，因为FPGA开发板内部的寄存器上电会自动置零。

Utape	lui x1,0xfffff auipc x5,0xfff	11111111111111111111000010110111 00000000111111111111001010010111		 出了一个问题，Rd信号变化的太快了，导致计算完第一条指令，存入reg的时候，Rd已经变成了下一条指令的目的寄存器  1.五级流水线每隔3个时钟周期，pc才更新一次 原因：取指阶段的pc刚开始是0，IF/ID流水线寄存器刚开始也是0，译码阶段也是0…… 这就导致前几个时钟周期内，EX阶段的pc+4的结果连续都是4，进而导致连续3个8，连续3个12…… 啊啊啊啊啊啊啊我就是到这里解决不来了我哭啊啊啊啊啊

访存指令	addi x1,x0,0x70
	slli x2,x1,8
	addi x1,x0,0x71
	add x2,x2,x1
	slli x2,x2,8
	addi x1,x0,0xf2
	add x2,x2,x1
	slli x2,x2,8
	addi x1,x0,0xf3
	add x2,x2,x1
	sw x2,0,x0
	sh x2,4,x0
	sb x2,7,x0
	lb x3,0,x0
	lb x4,1,x0
	lb x5,2,x0
	lb x6,3,x0
	lbu x7,0,x0
	lbu x8,1,x0
	lbu x9,2,x0
	lbu x10,3,x0
	lh x11,0,x0
	lh x12,2,x0
	lh x13,4,x0
	lhu x14,0,x0
	lhu x15,2,x0
	lhu x16,4,x0
	lw x17,0,x0
		00000111000000000000000010010011
		00000000100000001001000100010011
		00000111000100000000000010010011
		00000000000100010000000100110011
		00000000100000010001000100010011
		00001111001000000000000010010011
		00000000000100010000000100110011
		00000000100000010001000100010011
		00001111001100000000000010010011
		00000000000100010000000100110011
		00000000001000000010000000100011
		00000000001000000001001000100011
		00000000001000000000001110100011
		00000000000000000000000110000011
		00000000000100000000001000000011
		00000000001000000000001010000011
		00000000001100000000001100000011
		00000000000000000100001110000011
		00000000000100000100010000000011
		00000000001000000100010010000011
		00000000001100000100010100000011
		00000000000000000001010110000011
		00000000001000000001011000000011
		00000000010000000001011010000011
		00000000000000000101011100000011
		00000000001000000101011110000011
		00000000010000000101100000000011
		00000000000000000010100010000011
			这个是register
			 
			
			这个是ram
			 
				搞到这里忘了该看什么指标笑死
				 
				data_memory.v - 记事本
				文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
				input clk; //时钟信号
				input rst_n; //读出的指令
				input W_en; //写使能
				input R_en; //读使能
				input [31:0]addr; //读写地址
				input [2:0]RW_type; //读写类型
				input [31:0]din; //要写入存储器的数据
				output [31:0]dout; //要从数据存储器读出的数据
				reg [31:0]ram[255:0]; //数据存储器储存空间
				wire [31:0]Rd_data; //读的数据
				wire [31:0]Wr_data; //写的数据
				reg [31:0]Wr_data_B; //字节拼接
				wire [31:0]Wr_data_H; //半字拼接
				assign Rd_data=ram[addr[31:2]]; //读基准?????????

逻辑运算指令	addi x1,x0,0b00110011
	addi x2,x0,0b11111100
	and x3,x1,x2
	or x4,x1,x2
	xor x5,x1,x2
	andi x6,x1,0b11111100
	ori x7,x1,0b11111100
	xori x8,x1,0b11111100
		00000011001100000000000010010011
		00001111110000000000000100010011
		00000000001000001111000110110011
		00000000001000001110001000110011
		00000000001000001100001010110011
		00001111110000001111001100010011
		00001111110000001110001110010011
		00001111110000001100010000010011
				

移位运算指令	addi x1,x0,0xff
	addi x2,x0,4
	sll x3,x1,x2
	srl x4,x1,x2
	sra x5,x1,x2
	
	addi x6,x0,-0xff
	sll x7,x6,x2
	srl x8,x6,x2
	sra x9,x6,x2
	
	slli x11,x1,4
	srli x12,x1,4
	srai x13,x1,4
	
	slli x11,x6,4
	srli x12,x6,4
	srai x13,x6,4
		00001111111100000000000010010011
		00000000010000000000000100010011
		00000000001000001001000110110011
		00000000001000001101001000110011
		01000000001000001101001010110011
		11110000000100000000001100010011
		00000000001000110001001110110011
		00000000001000110101010000110011
		01000000001000110101010010110011
		00000000010000001001010110010011
		00000000010000001101011000010011
		01000000010000001101011010010011
		00000000010000110001010110010011
		00000000010000110101011000010011
		01000000010000110101011010010011
				

小于置一指令	addi x1,x0,-1
	addi x2,x0,-2
	addi x3,x0,3
	addi x4,x0,4
	slt x5,x1,x2
	slt x6,x2,x1
	slt x7,x3,x4
	slt x8,x4,x3
	sltu x9,x1,x3
	sltu x10,x3,x1
	sltu x11,x1,x2
	slti x12,x3,-3
	slti x13,x3,4
	slti x14,x2,-1
	slti x15,x2,-4
	sltiu x16,x3,-3
	sltiu x17,x3,4
	sltiu x18,x2,-1
	sltiu x19,x2,-4
		11111111111100000000000010010011
		11111111111000000000000100010011
		00000000001100000000000110010011
		00000000010000000000001000010011
		00000000001000001010001010110011
		00000000000100010010001100110011
		00000000010000011010001110110011
		00000000001100100010010000110011
		00000000001100001011010010110011
		00000000000100011011010100110011
		00000000001000001011010110110011
		11111111110100011010011000010011
		00000000010000011010011010010011
		11111111111100010010011100010011
		11111111110000010010011110010011
		11111111110100011011100000010011
		00000000010000011011100010010011
		11111111111100010011100100010011
		11111111110000010011100110010011