



单周期RISCV处理器介绍

薛臻

2021/7/12

■ RISC-V 64I 指令集架构

■ 单周期 CPU 代码框架

■ 一条指令的执行过程

■ 经典五级流水线

■ 单周期代码使用指南

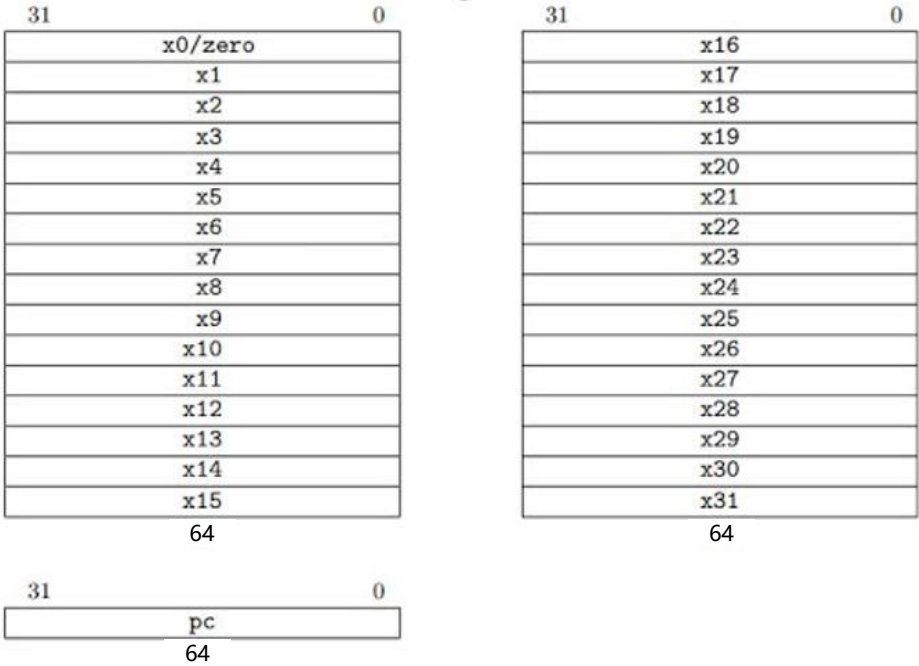


RISCV64I指令集架构

- RISC指令集
- 六种指令格式
- RISCV64I寄存器
 - **PC**: 程序计数器, 存放下一条指令的地址
 - **32个通用寄存器**, 包括1个值恒为0的x0寄存器
- 小端存储

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0	
funct7				rs2		rs1	funct3		rd			opcode		R-type	
imm[11:0]						rs1	funct3		rd			opcode		I-type	
imm[11:5]				rs2		rs1	funct3		imm[4:0]			opcode		S-type	
imm[12]	imm[10:5]			rs2		rs1	funct3		imm[4:1]	imm[11]		opcode		B-type	
imm[31:12]									rd			opcode		U-type	
imm[20]	imm[10:1]			imm[11]		imm[19:12]			rd			opcode		J-type	

RISCV指令格式



RISCV寄存器

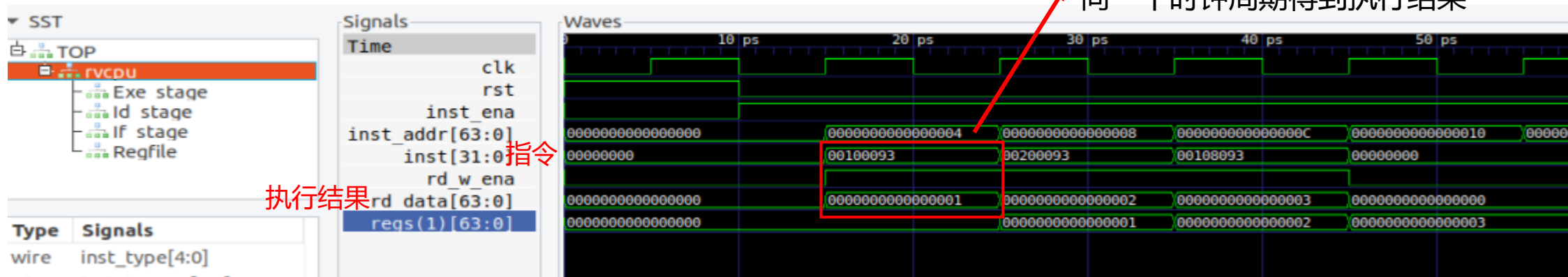


单周期CPU代码框架

- 取指阶段：从内存中获取当前PC中对应的指令，同时PC+4
- 译码阶段：解析指令，获得源/目的操作数等信息
- 执行阶段：进行指令对应的运算等

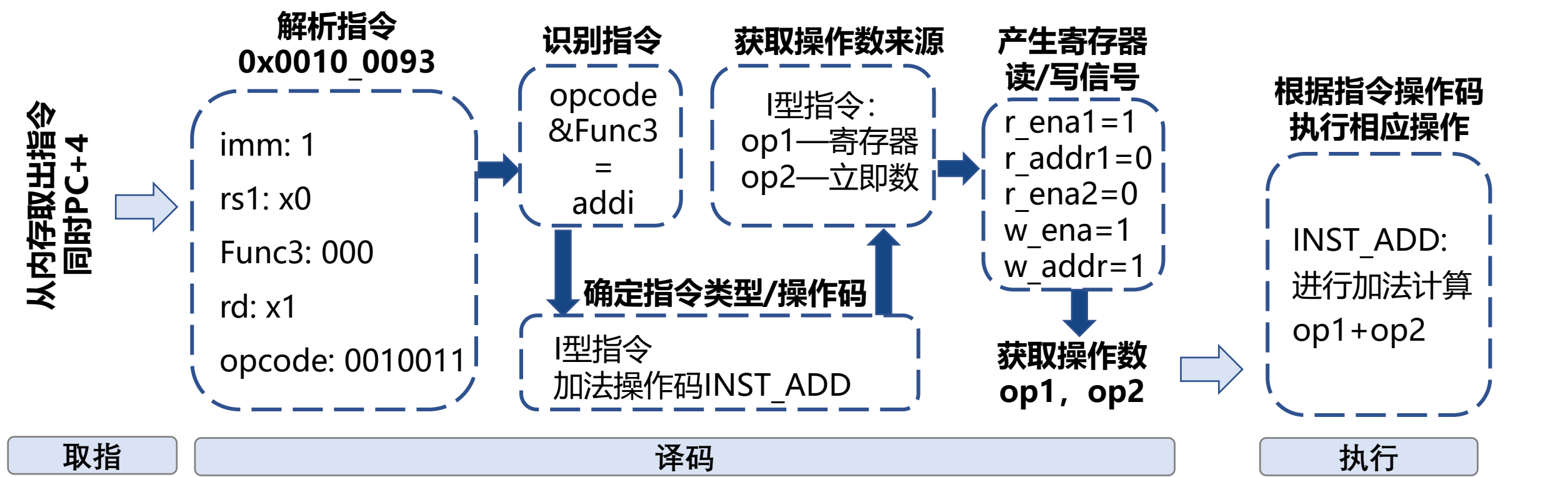
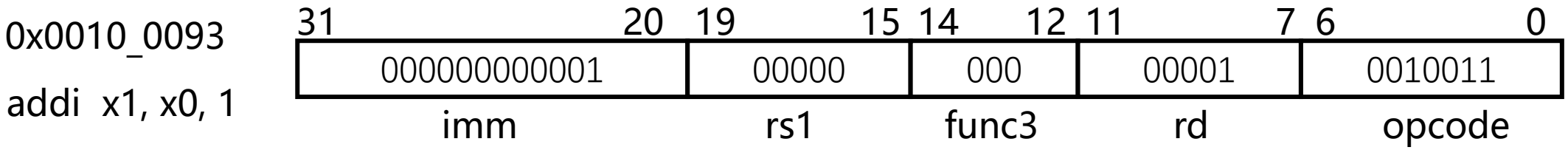
Single cycle riscv cpu

```
├── defines.v           //存放一些常用的变量
├── exe_stage.v         //执行阶段
├── id_stage.v          //译码阶段
├── if_stage.v          //取指阶段
├── inst.bin            //指令文件
├── regfile.v           //riscv寄存器
├── rvcpu-test.cpp      //verilator的仿真文件
└── rvcpu.v             //riscvCPU
```





一条指令的执行过程——以addi为例





一条指令的执行过程——以addi为例

```
always@( posedge clk )
begin
    if( rst == 1'b1 )
    begin
        pc <= `ZERO_WORD ;
    end
    else
    begin
        pc <= pc + 4;
    end
end
```

取指

```
assign opcode = inst[6 : 0];
assign rd      = inst[11 : 7];
assign func3   = inst[14 : 12];
assign rs1     = inst[19 : 15];
assign imm     = inst[31 : 20];
```

译码：解析指令

```
wire inst_addi = ~opcode[2] & ~opcode[3] & opcode[4] & ~opcode[5] & ~opcode[6]
                & ~func3[0] & ~func3[1] & ~func3[2];
```

译码：识别指令

译码：获取操作数

```
assign op1 = ( rst == 1'b1 ) ? 0 : ( inst_type[4] == 1'b1 ? rs1_data : 0 );
assign op2 = ( rst == 1'b1 ) ? 0 : ( inst_type[4] == 1'b1 ? { {52{imm[11]}} ,
```

```
assign rs1_r_ena = ( rst == 1'b1 ) ? 0 : inst_type[4];
assign rs1_r_addr = ( rst == 1'b1 ) ? 0 : ( inst_type[4] == 1'b1 ? rs1 : 0 );
assign rs2_r_ena = 0;
assign rs2_r_addr = 0;
```

译码：产生寄存器读/写信号

```
assign rd_w_ena = ( rst == 1'b1 ) ? 0 : inst_type[4];
assign rd_w_addr = ( rst == 1'b1 ) ? 0 : ( inst_type[4] == 1'b1 ? rd : 0 );
```

```
// arith inst: 10000; logic: 01000;
// load-store: 00100; j: 00010; sys: 000001
assign inst_type[4] = ( rst == 1'b1 ) ? 0 : inst_addi;

assign inst_opcode[0] = ( rst == 1'b1 ) ? 0 : inst_addi;
assign inst_opcode[1] = ( rst == 1'b1 ) ? 0 : 0;
assign inst_opcode[2] = ( rst == 1'b1 ) ? 0 : 0;
assign inst_opcode[3] = ( rst == 1'b1 ) ? 0 : 0;
assign inst_opcode[4] = ( rst == 1'b1 ) ? 0 : inst_addi;
assign inst_opcode[5] = ( rst == 1'b1 ) ? 0 : 0;
assign inst_opcode[6] = ( rst == 1'b1 ) ? 0 : 0;
assign inst_opcode[7] = ( rst == 1'b1 ) ? 0 : 0;
```

译码：确定
类型/操作码

```
case( inst_opcode )
    `INST_ADD: begin rd_data = op1 + op2; end
    default:   begin rd_data = `ZERO_WORD; end
endcase
```

执行

- 单周期RISCV CPU：在一个周期内得到执行结果
- 取出指令后，其译码、执行过程均为组合逻辑
- 思考：如果是五级流水线，得到执行结果需要几个时钟周期？需要怎样添加时序逻辑？



经典五级流水线

- **IF**: 取指 将指令从存储器中读出
- **ID**: 译码 翻译指令, 得到源/目的操作数的相关信息
- **EX**: 执行 执行指令对应的操作
- **MEM**: 访存 对存储器进行读/写
- **WB**: 写回 将执行结果写入通用寄存器

流水线: 通过提高CPU中**各个部件的利用率**, 提高其工作效率

Instr No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

指令1, 2, 3
顺序执行: $5 * 3 = 15$ 个周期
流水线执行: 7个周期



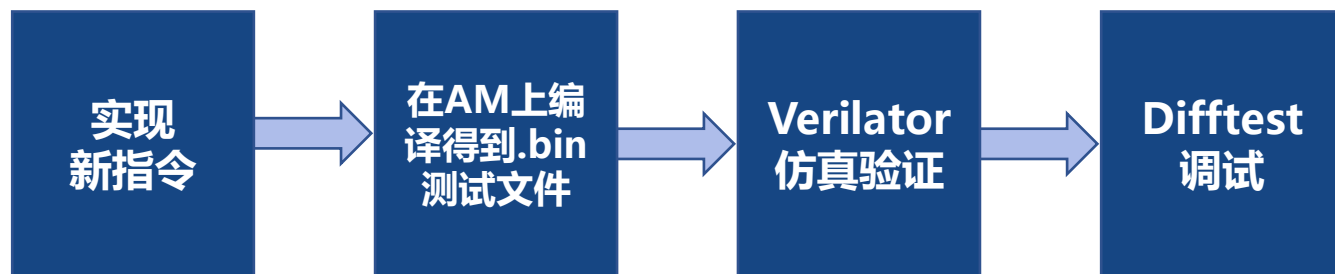
单周期代码使用指南

- 只实现了取指、译码和执行阶段，**请完成访存和写回阶段**
- 添加时序逻辑，**实现五级流水**
- **思考：**
 - **数据相关：** RAW（写后读），WAR（读后写），WRW（写后写）哪种需要关注？
 - **控制相关：** 分支指令如何处理？
 - **流水线暂停（stall）、流水线删除（kill）** 怎样实现？
- 单周期代码仅供参考

实现访存、写回阶段后，将寄存器写使能、写地址和写数据逐级下传到写回阶段

```
assign rd_w_ena    = ( rst == 1'b1 ) ? 0 : inst_type[4];  
assign rd_w_addr   = ( rst == 1'b1 ) ? 0 : ( inst_type[4] == 1'b1 ? rd   : 0 );  
                                                           id_stage.v
```

```
output reg  [`REG_BUS]rd_data    exe_stage.v
```



功能验证流程



谢谢

薛臻

2021/7/12