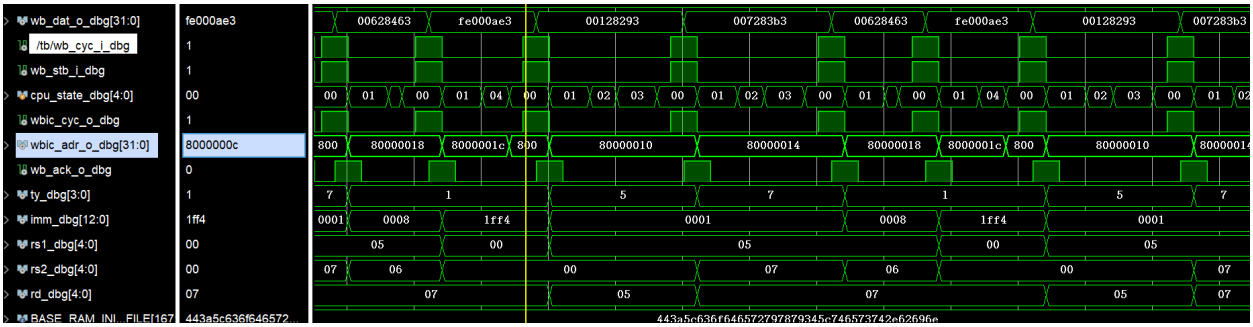
计组实验5 - 实验报告

于新雨 计25 2022010841

跳转时，写入内存和写入串口时的波形图和对应的指令译码情况

跳转时

以 `beq zero, zero, loop` 为例

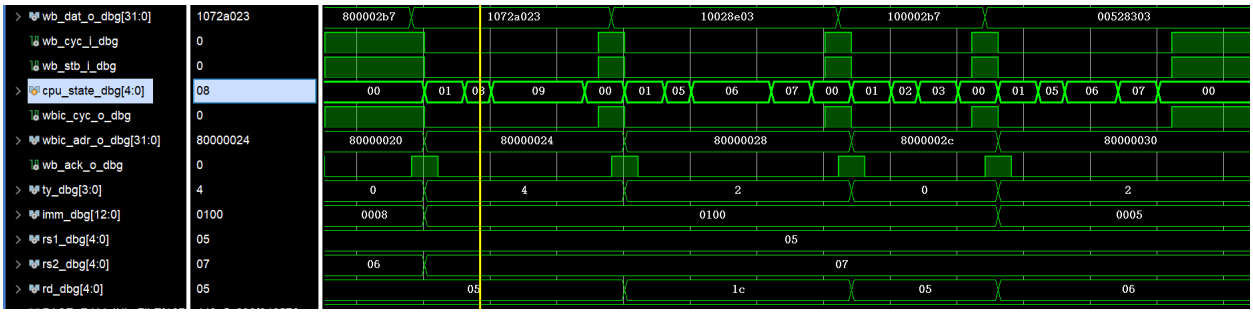
- 波形图
  - 
  - 注：因为在通过 oj 后接着实现了 icache 所以图中为输出的 debug 信号，如图，`wbic_adr_o_dbg` 代表向 icache 请求的指令地址，即当前指令地址
- 指令译码情况
  - 定义类型

```
typedef enum logic [3:0]{
LUI,        // 0
BEQ,        // 1
LB,         // 2
SB,        // 3
SW,        // 4
ADDI,      // 5
ANDI,      // 6
ADD,        // 7
UNK  // unknown 8
}instr_type;
```

  - 如图，ty 识别为1，即是 `BEQ` 类型的指令，imm 是 0x1ff4 即 -12，rs1 是 0x0，rs2 是 0x0

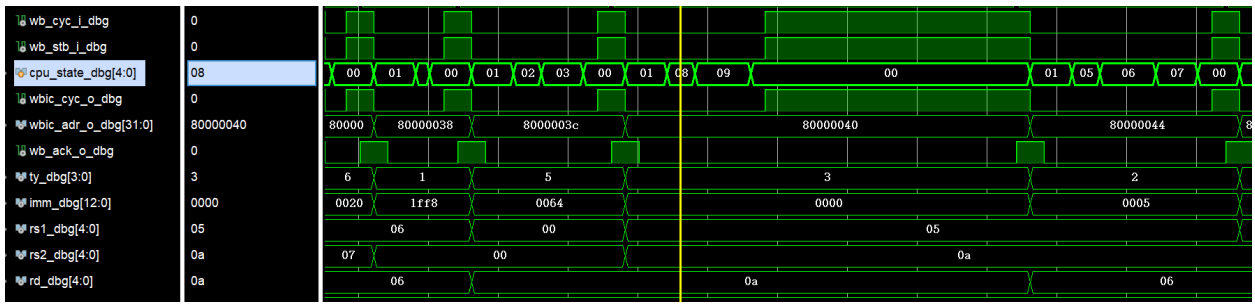
写入内存时

以 `sw t2, 0x100(t0)` 为例

- 波形图
  - 
- 译码
  - 可见 ty 为 4，即 `SW` 类型的指令，imm 为 0x100，rs1 为 0x5，rs2 为 0x7 rd 为 0x5

写入串口时

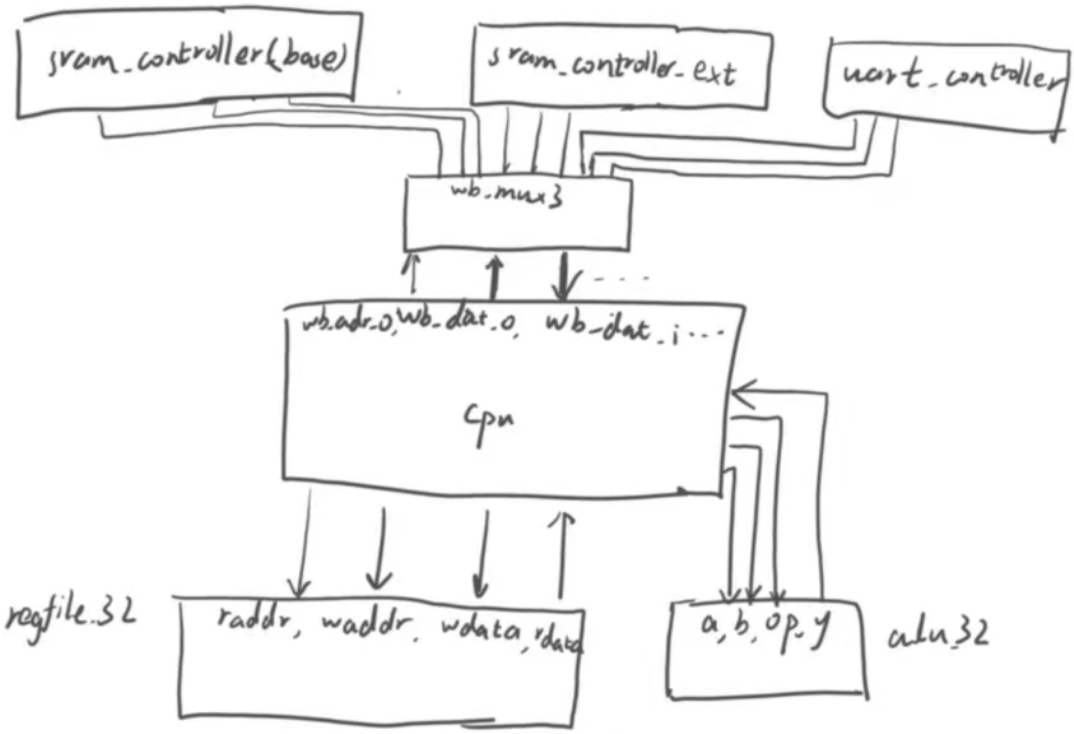
以 `sb a0, 0(t0)` 为例

- 波形图
  - 

- 译码
  - 可见 ty 为 3，即 `SB` 类型的指令，imm 为 0，rs1 为 0x5，rs2 为 0xa，rd 为 0xa

内容要求

cpu 结构图



来自 HUAWEI MatePad Paper

信号表和状态转移表

信号表

信号名	信号类型	信号描述
clk	input	时钟信号
rst_n	input	复位信号

信号名	信号类型	信号描述
wb_cyc_o	output	wishbone 总线周期信号
wb_stb_o	output	wishbone 总线使能信号
wb_ack_i	input	wishbone 总线应答信号
wb_adr_o	output	wishbone 总线地址信号
wb_dat_o	output	wishbone 总线数据信号
wb_dat_i	input	wishbone 总线数据输入信号
wb_sel_o	output	wishbone 总线选择信号
wb_we_o	output	wishbone 总线写使能信号
alu_a	output	alu 输入 操作数1
alu_b	output	alu 输入 操作数2
alu_op	output	alu 操作码
alu_y	input	alu 输出
raddr_a	output	register file 读地址1
raddr_b	output	register file 读地址2
waddr	output	register file 写地址
wdata	output	register file 写数据
we	output	register file 写使能
rdata_a	input	register file 读数据1
rdata_b	input	register file 读数据2
pc_reg	reg	当前指令地址(IF)或者下一条指令地址(其他段)
pc_now_reg	reg	当前指令地址
instr_reg	reg	当前指令
state	reg	状态机
regfile_state	reg	寄存器文件状态机（因为 regfile 不是当前周期返回）
ty	reg	指令类型
imm	reg	立即数
lui_imm	reg	长立即数
rd	reg	目的寄存器
rs1	reg	源寄存器1
rs2	reg	源寄存器2
rs1_val	reg	源寄存器1的值
rs2_val	reg	源寄存器2的值
rd_val	reg	目的寄存器的值
exe_arith_done	reg	算术运算是否完成
shift_val	reg	非对齐访问右移位数
sram_addr_tmp	reg	临时算出的 sram 地址
exe_beq_done	reg	beq 是否完成

信号名	信号类型	信号描述
exe_mem_done	reg	load/store 是否完成
reg_write_state	reg	写回寄存器的状态
is_split	reg	非对齐访问是否需要分两次写入（sw 的场景）

Arith (ADDI,ANDI,LUI,ADD)

STATE	wb_addr	wb_cyc	rf_addra	rf_addrb	alu_a	alu_b	op	rf_we	rf_wdata	rf_waddr
STATE_IF	pc	1	unk	unk	unk	unk	unk	0	unk	unk
STATE_ID	unk	0	rs1	rs2	unk	unk	unk	0	unk	unk
STATE_EXE	unk	0	unk	unk	rs1_val	rs2_val	alu_op	0	unk	unk
STATE_WRITE_ARITH	unk	0	unk	unk	unk	unk	unk	1	alu_y	rd

BEQ

STATE	wb_addr	wb_cyc	rf_addra	rf_addrb	rs1_val	rs2_val	pc
STATE_IF	pc	1	unk	unk	unk	unk	unk
STATE_ID	unk	0	rs1	rs2	rs1_val	rs2_val	unk
STATE_EXE	unk	0	unk	unk	unk	unk	if rs1_val == rs2_val then newpc

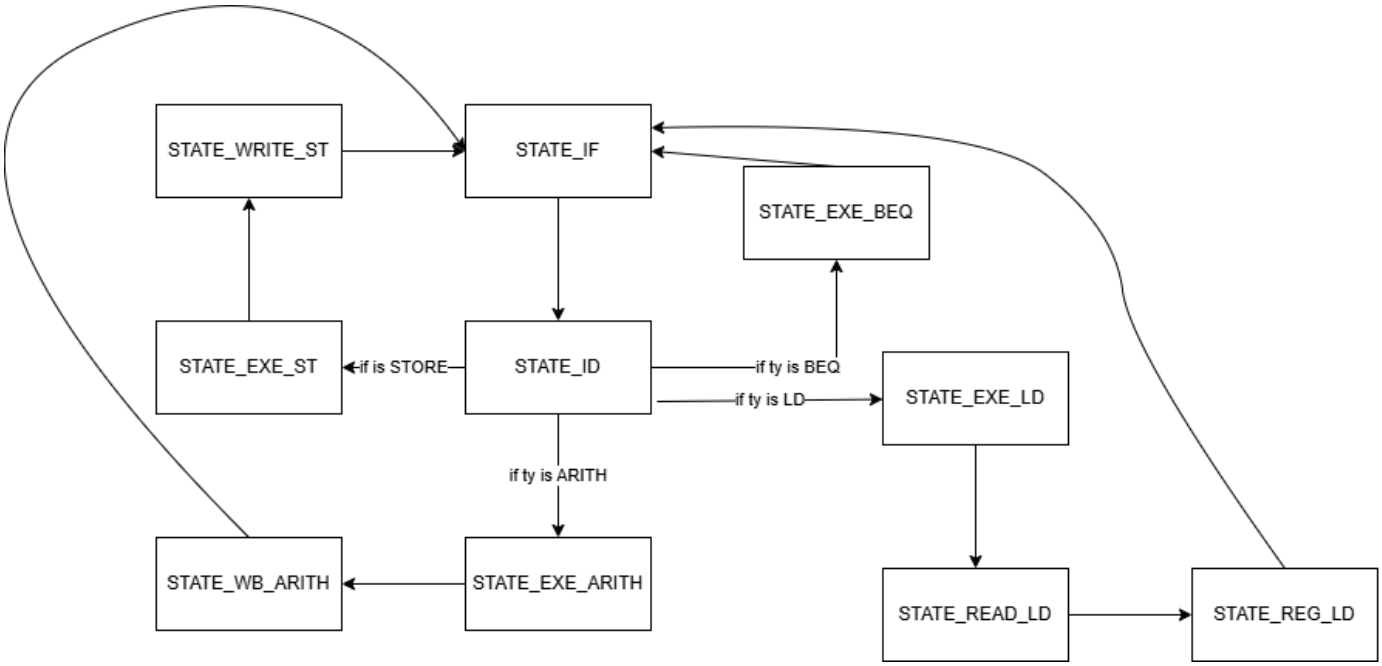
Load

STATE	wb_addr	wb_cyc	rf_addra	rf_addrb	alu_a	alu_b	op	rf_we	rf_wdata	rf_waddr
STATE_IF	pc	1	unk	unk	unk	unk	unk	0	unk	unk
STATE_ID	unk	0	rs1	rs2	unk	unk	unk	0	unk	unk
STATE_EXE_LD	unk	0	unk	unk	rs1_val	imm	unk	0	unk	unk
STATE_READ_LD	imm + rs1_val	1	unk	unk	unk	unk	unk	0	0	0
STATE_REG_LD	unk	0	unk	unk	unk	unk	unk	1	sram_data	rd

Store

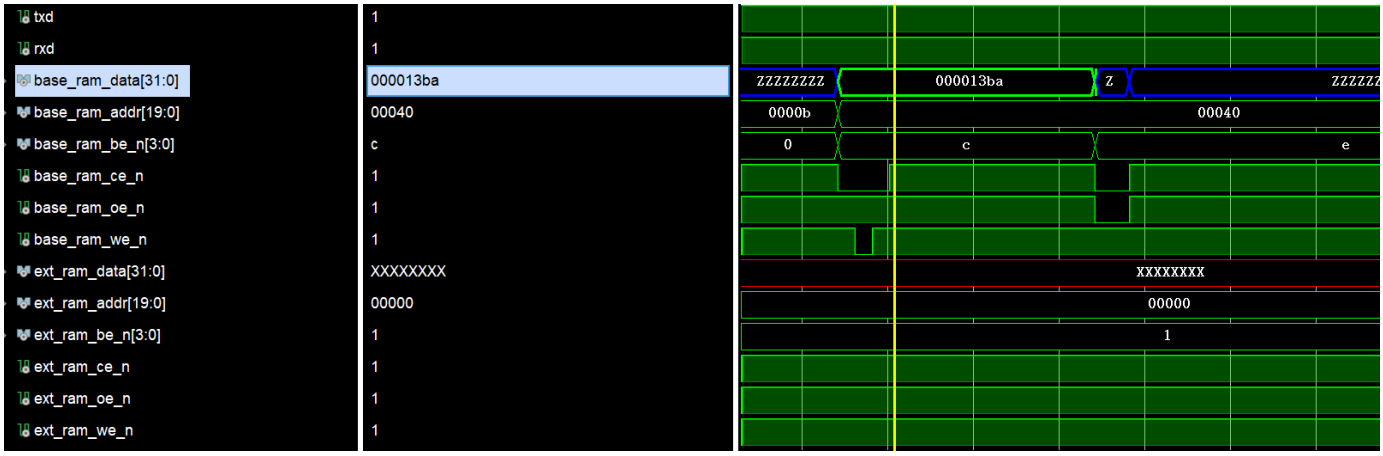
STATE	wb_addr	wb_cyc	rf_addra	rf_addrb	alu_a	alu_b	op	rf_we	rf_wdata	rf_waddr	wb_dat_i
STATE_IF	pc	1	unk	unk	unk	unk	unk	0	unk	unk	unk
STATE_ID	unk	0	rs1	rs2	unk	unk	unk	0	unk	unk	unk
STATE_EXE_ST	unk	0	unk	unk	rs1_val	imm	unk	0	unk	unk	unk
STATE_WRITE_ST	imm + rs1_val	1	unk	unk	unk	unk	unk	0	0	0	rs2_val

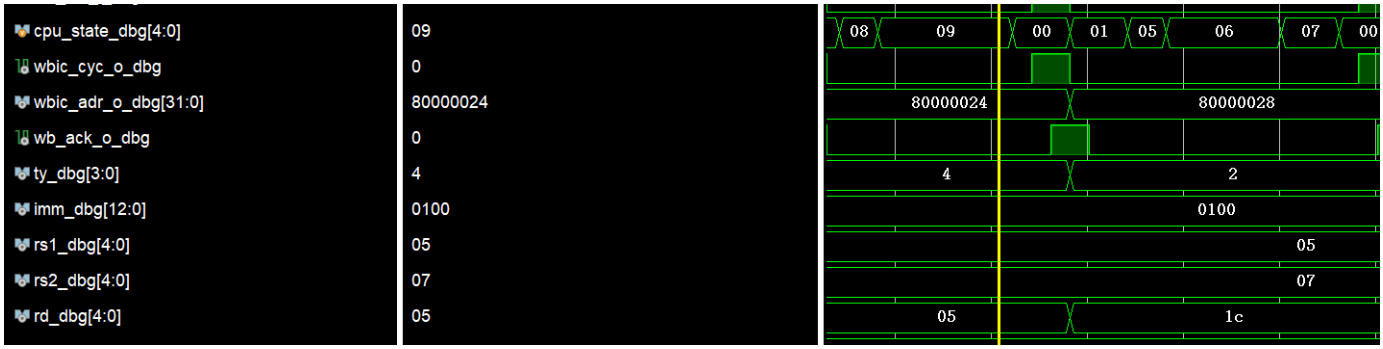
状态转移表



内存数据

在 `sw t2, 0x100(t0)` 后增加 `lb t3, 0x100(t0)` 查看 sram 返回值





可见从 base_ram 返回了计算的正确结果即 5050

串口输出

```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:09 . Memory (MB): peak = 1462.703 ; gain = 0.000
run all
[121572 ns]: uart received 0x64, ASCII: d
[215973 ns]: uart received 0x6f, ASCII: o
[311459 ns]: uart received 0x6e, ASCII: n
[403689 ns]: uart received 0x65, ASCII: e
[495919 ns]: uart received 0x21, ASCII: !
close [ open D:/coderyxy4/cod24-xy-yu22/thinpad_top.srcs/sources_1/new/dcache.sv w ]
add_files D:/coderyxy4/cod24-xy-yu22/thinpad_top.srcs/sources_1/new/dcache.sv
```

思考题

- 流水线 CPU 中，用于 branch 指令的比较器既可以放在 ID 阶段，也可以放在 EXE 阶段。放在这两个阶段分别有什么优缺点？
- 放到 ID 阶段
  - 优点
    - 早发现分支，可以提前进行分支预测，减少分支延迟
  - 缺点
    - 可拓展性差，如果分支条件需要依赖 EXE 阶段计算的结果（例如计算某些寄存器值），则无法在 ID 阶段直接做出跳转决策
    - 代码实现上 ID 阶段本身组合逻辑较复杂，可能增大 ID 段时延
- 放到 EXE 阶段
  - 优点
    - 在 EXE 阶段执行分支条件比较符合流水线逻辑，减少了 ID 阶段的复杂度
    - 分支条件可以依赖 EXE 阶段的计算结果，可以设计更加复杂 更具拓展性的指令
  - 缺点
    - 增加分支延迟，直到 EXE 阶段才能确定跳转条件，延迟了分支的决策，可能导致更多的气泡和性能损失
    - 需要等待 EXE 阶段完成后才能更新 PC 寄存器，从而可能影响整个流水线的吞吐量