架构实验4

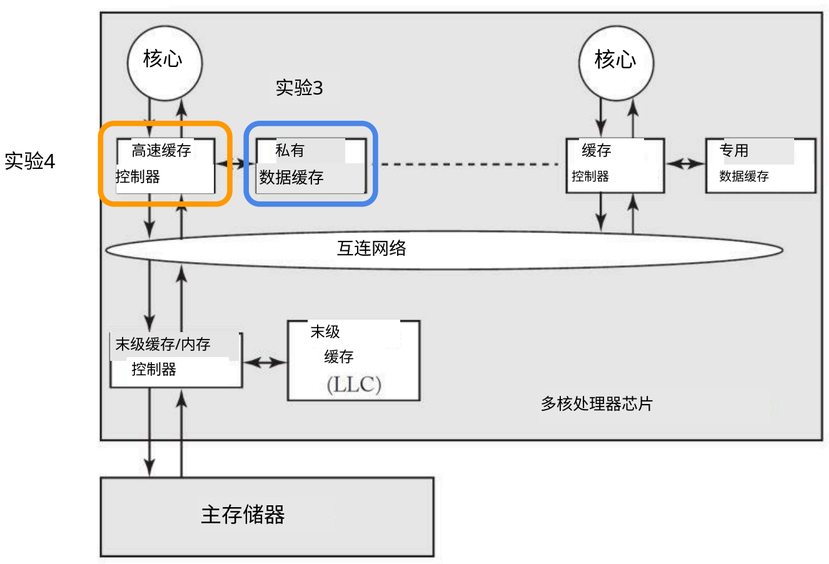
Pipelined CPU with Cache

Chenlu Miao 11/2022

概述

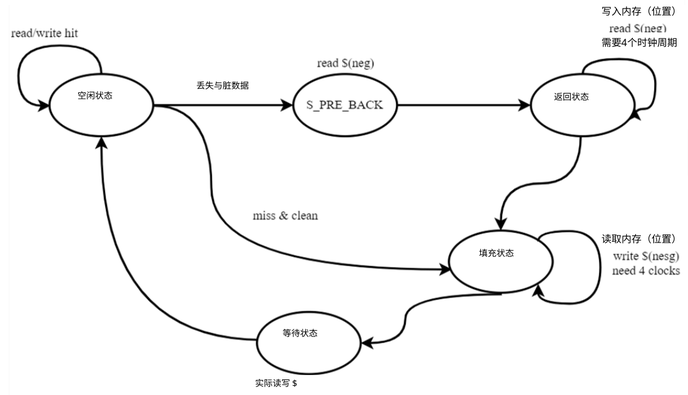
- 缓存概述
- 缓存控制器接口
- 缓存控制器有限状态机
- 代码：缓存控制器
- 仿真

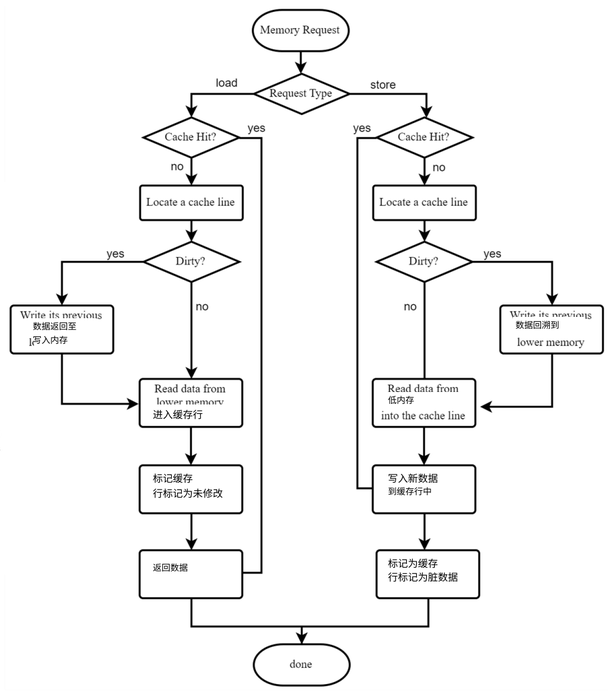
缓存概述



缓存控制器接口

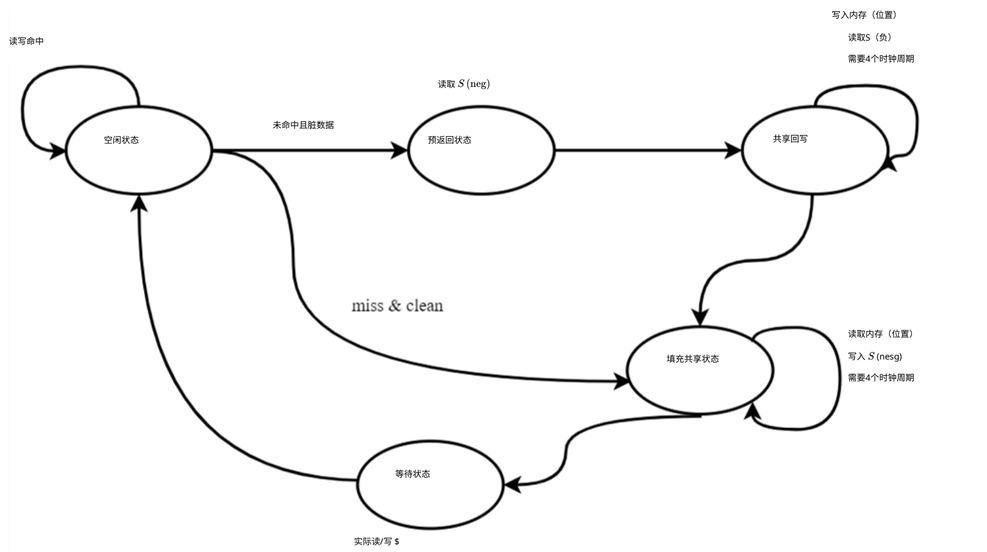
缓存控制器有限状态机



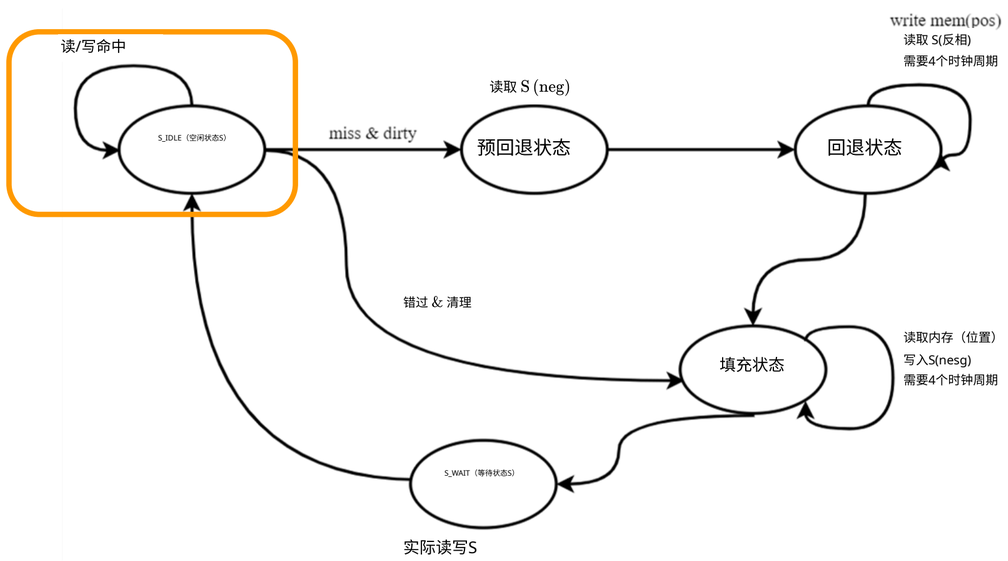


缓存控制器有限状态机

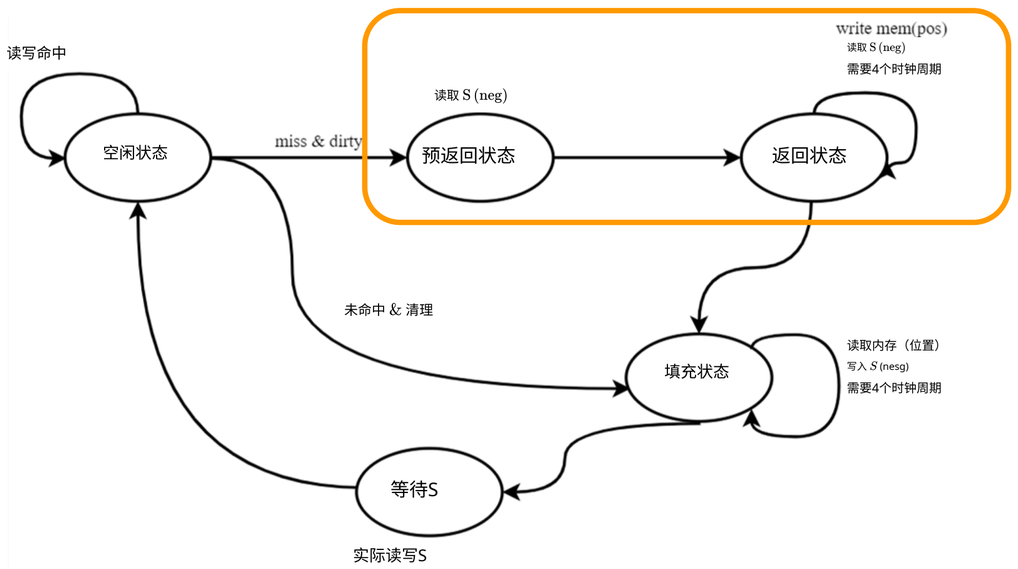
- 空闲状态
- 预备份状态
- 回写状态
- 填充共享状态
- 等待状态



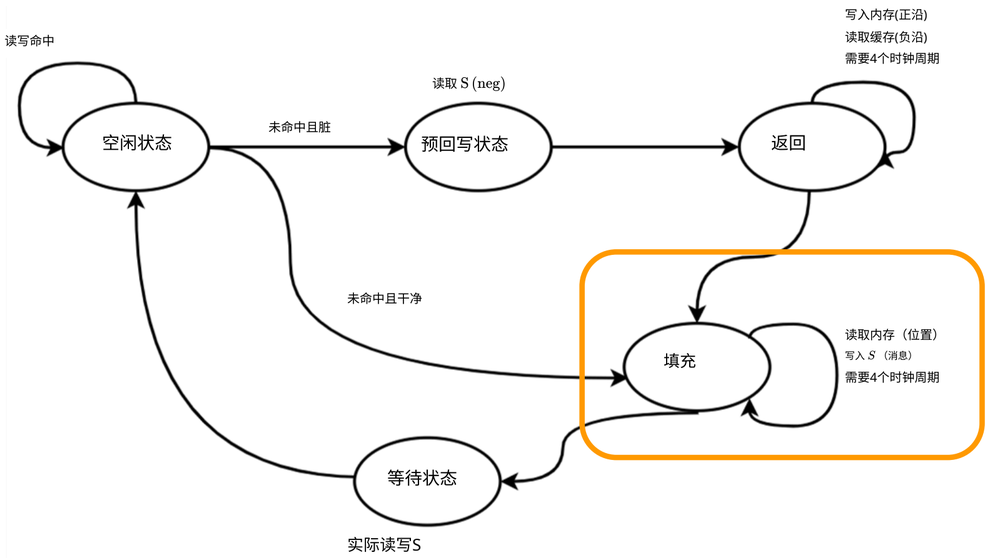
缓存控制器有限状态机 - 命中



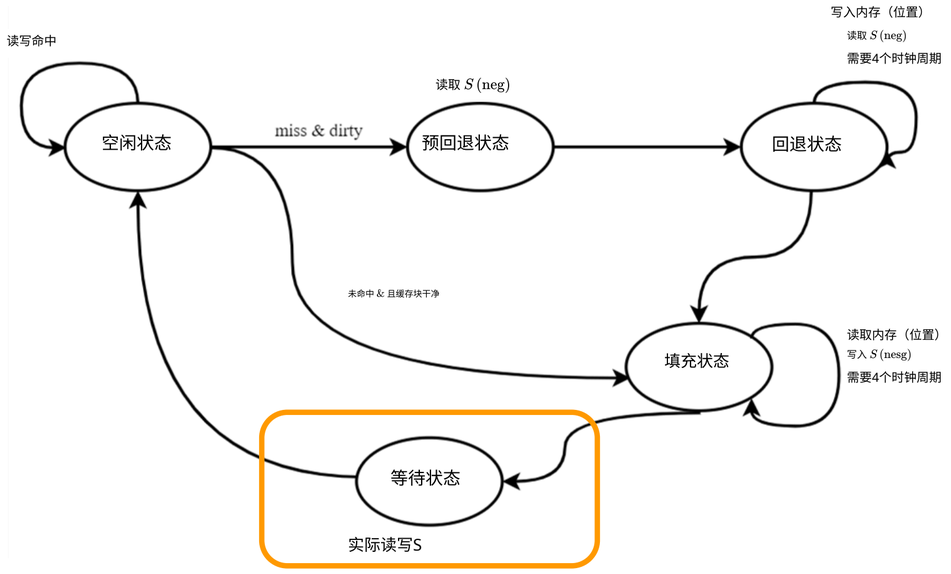
缓存控制器有限状态机 - 未命中且脏数据



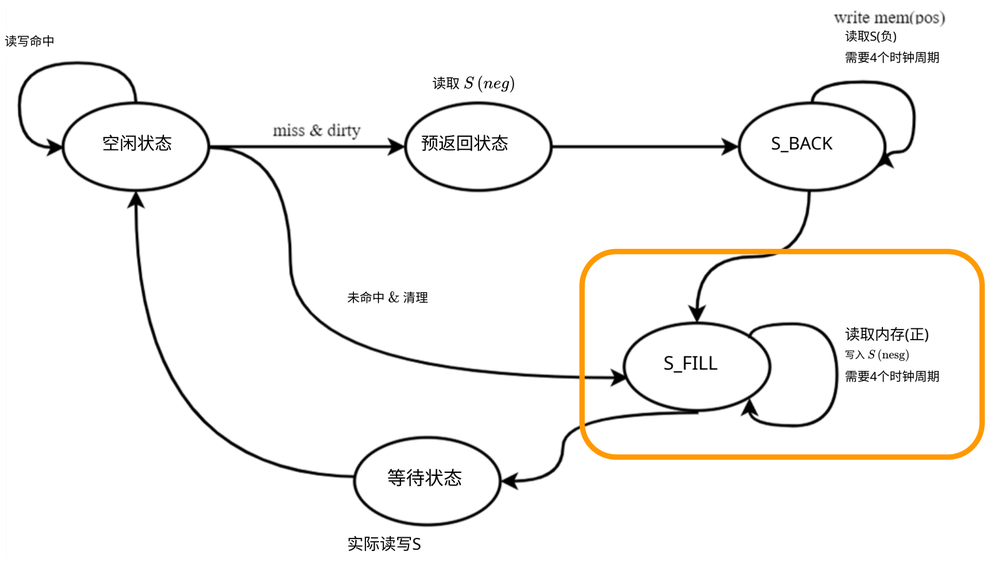
缓存控制器有限状态机 - 未命中且脏数据



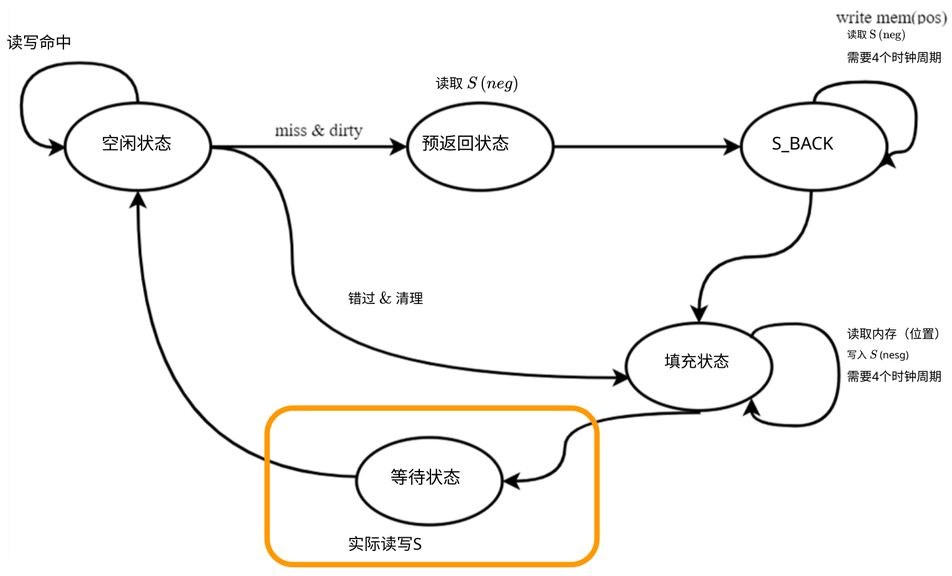
缓存控制器有限状态机 - 未命中且脏数据



缓存控制器有限状态机 - 未命中且缓存块干净



缓存控制器有限状态机 - 未命中且缓存块干净



代码：缓存控制器

本地参数
    S_IDLE = 0,
    S_PRE_BACK = 1,
    S_BACK = 2,
    S_FILL = 3,
    等待状态（S_WAIT）等于4;
    [2:0]状态（state）等于0;
    全局（g）[2:0]下一状态（next_state）等于0;
    全局（g）[元素字宽（ELEMENT_WORDS_WIDTH） - 1:0]字数（word_count）等于0;
寄存器（reg）[元素字宽（ELEMENT_WORDS_WIDTH） - 1:0]下一字数（next_word_count）等于0;

始终在（时钟上升沿（posedge clk））开始
    如果 (rst) 则开始
        状态 <= S_IDLE;
        字计数 <= 2'b00;
    结束
    否则开始
        状态 <= 下一状态;
        单词计数 <= 下一个单词计数;
    结束
结束

代码：缓存控制器

始终 â (*) 开始
    如果 (rst) 则开始
        下一状态 = S_IDLE;
        下一个字计数 = 2'b00;
    结束
    否则开始
        根据 (状态) 进行选择
            空闲状态（S_IDLE）：开始
                如果 (en_r 或 en_w) 成立
                    如果 (缓存命中)
                        下一状态 = ??;
                    否则，如果 (缓存有效且缓存已脏)
                        下一状态 = ??;
                    否则
                        下一状态 = ??;
                结束
                下一个单词计数 = 2'b00;
            结束

预回退状态（S_PRE_BACK）：开始
        下一状态 = ??;
        下一个单词计数 = 2位二进制数00;
    结束

    状态S_BACK：开始
        如果 (内存确认信号输入且单词计数 == ...)
            下一状态 = ??;
        否则
            下一状态 = ??;
        如果 (内存确认信号输入)
            下一个字计数 = ??;
        否则
            下一个字计数 = 当前字计数;
    结束

代码：缓存控制器

S_FILL：开始
    如果 (内存确认信号且单词计数 == ...)
        下一状态 = ??;
    否则
        下一状态 = ??;
    如果 (内存确认信号输入)
        下一个字计数 = ??;
    否则
        下一个单词计数 = 单词计数;
结束

S_WAIT: 开始
    下一个状态 = ??;
    下一个单词计数 = 2位二进制数00;
结束

代码: 缓存控制器

始终在 (*) 开始
    根据状态情况
        空闲状态、等待状态: 开始
            缓存地址 = 读写地址;
            缓存加载 = 读使能;
            缓存编辑 = 英文单词;
            缓存存储 = 1位二进制0;
            缓存无符号字节/半字/字 = 无符号字节/半字/字;
            缓存数据输入 = 写入数据;
        结束

    S_BACK, S_PRE_BACK: 开始
        缓存地址 = ...
        缓存加载 = 1'b0;
        缓存编辑 = 1'b0;
        缓存存储 = 1位二进制0;
        缓存上半字节高位字 = 3位二进制010;
        缓存数据输入 = 32位二进制0;
    结束
    S_FILL: 开始
        缓存地址 = ...
        缓存加载 = 1'b0;
        缓存编辑 = 1位二进制0;
        缓存存储 = 内存确认信号输入;
        缓存上边界高度宽度 = 3位二进制数010;
        缓存输入数据 = 内存数据输入;
    结束
    结束条件判断
结束
将 data_r 赋值为 cache_dout;

代码：缓存控制器

始终 @ (*) 开始

根据 (下一状态) 情况

空闲状态 (S_IDLE)、预回退状态 (S_PRE_BACK)、等待状态 (S_WAIT)：开始

内存片选输出 (mem_cs_o) = 1'b0;

内存写使能输出 (mem_we_o) = 1'b0;

内存地址输出信号 = 32位二进制0;

结束

状态S_BACK：开始

内存片选输出信号 = 1位二进制1;

内存写使能输出信号 = 1位二进制1;

内存地址输出信号 = ...

结束

S_FILL：开始

mem_cs_o = 1'b1;

mem_we_o = 1'b0;

mem_addr_o = ...

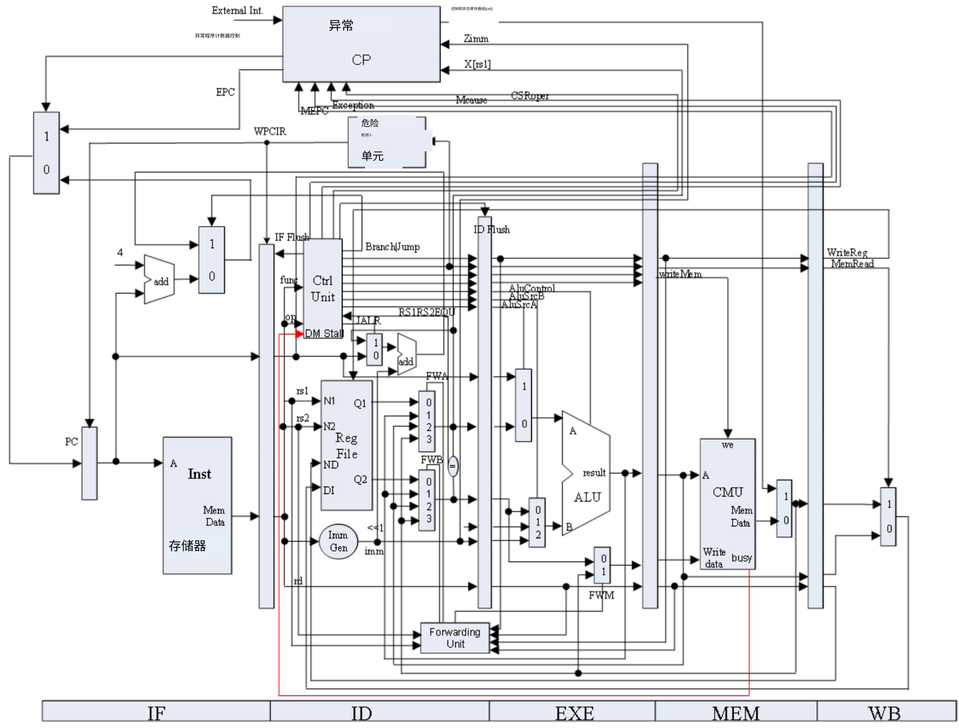
结束

结束情况

结束

将mem_data_o赋值为cache_dout;

带缓存的流水线 CPU



ROM

NO.	指令	地址	标签	ASM	注释
0	00000013	0	____开始:	将x0寄存器的值加0后存回x0	
1	01c00083	4		从x0寄存器的值加上0x01C的地址处加载一个字节到x1寄存器	#0x1C地址处为FOFOFOFO #FFFFFFF0未命中，读取 0x010~0x01C设置为第0行
2	01c01103	8		加载半字到x2，地址为0x01C(x0)	#FFFFF0F0 命中
3	01c02183	C		加载字到x3，地址为0x01C(x0)	#F0F0F0F0 命中
4	01c04203	10		加载无符号字节到x4，地址为0x01C(x0)	#000000F0 命中
5	01c05283	14		lhu x5, 0x01C(x0)	#0000F0F0 命中
6	21002003	18		lw x0, 0x210(x0)	#未命中，将 0x210~0x21C 读入缓存集 1 行 1
7	abcde0b7	1C		lui x1 0xABCDE	
8	71c08093	20		将0x71C加到x1寄存器的值上	#x1寄存器的值为0xABCDE71C
9	00100023	24		将x1寄存器的值存储到以x0寄存器的值为基地址、偏移量为0x0的内存地址中	#未命中，将0x000~0x00C的数据读入缓存组0的第0行
10	00101223	28		将x1寄存器的值以半字形式存储到以x0寄存器的值为基地址、偏移量为0x4的内存地址中	#命中
11	00102423	2C		将x1寄存器的值存储到以x0寄存器的值加0x8为地址的内存位置	#命中

ROM

NO.	指令	地址	标签	ASM	注释
12	20002303	30		lw x6, 0x200(x0)	#未命中，将0x200~0x20C读入缓存组0行1
13	40002383	34		lw x7, 0x400(x0)	#未命中，将0x000~0x00C写回内存，然后将0x400~40C读入缓存组0行0
14	41002403	38		从地址 (x0 + 0x410) 处加载 8 字节数据到寄存器 x8	#未命中，由于数据干净无需回写，将0x410~41C 读取到缓存集 1 的第 0 行
15	0ed06813	3c	循环:	将立即数 0xED 与寄存器 x0 进行或运算，结果存入寄存器 x16	#结束
16	ffdff06f	40		跳转到x0，循环	

RAM

NO.	数据	地址
0	00000000	0
1	00000004	4
2	00000008	8
3	0000000C	C
4	00000010	10
5	00000014	14
6	00000018	18
7	0000001C	1C
8	00000020	20
9	00000024	24
10	00000028	28
11	0000002C	2C

模拟

指令.v

寄存器 [39:0] 数据 [0:9]; 初始开始

数据[0] = 40'h0_2_00000004; // 读缺失

数据[1] = 40'h0_3_00000019; // 写缺失

data[2] = 40'h1_2_00000008; // 读命中 data[3] = 40'h1_3_00000014; // 写命中

data[4] = 40'h2_2_00000204; // 读未命中

data[5] = 40'h2_3_00000218; // 写未命中

data[6] = 40'h0_3_00000208; // 写命中

data[7] = 40'h4_2_00000414; // 读未命中 + 脏数据

data[8] = 40'h1_3_00000404; // 写未命中 + 干净数据

数据[9] = 40'h0; // 结束

结束

赋值

u_b_h_w = 数据[索引][38:36]，有效 = 数据[索引][33]，写入 = 数据[索引][32]，地址 = 数据[索引][31:0];

1+17

$1+17$

1

1

1+17

$1+17$

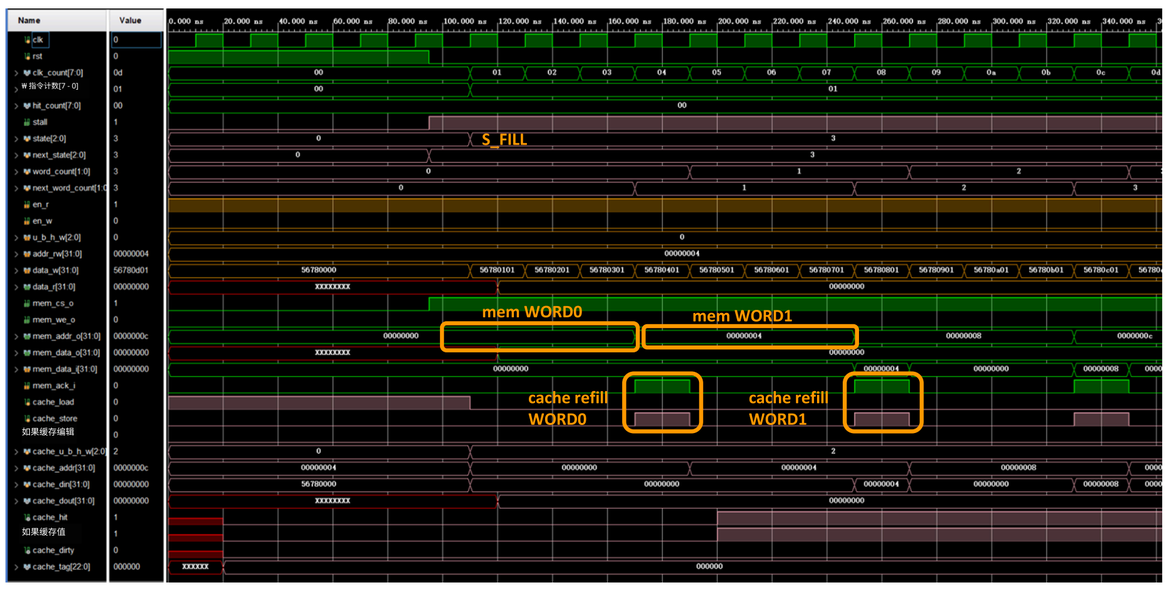
1

$1+17+17$

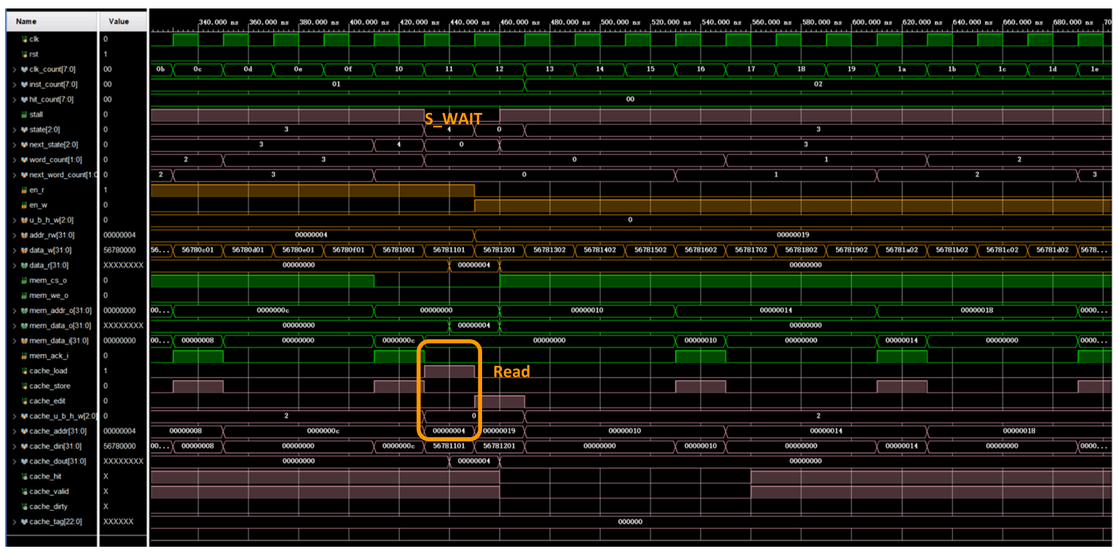
$1+17$

总计: 128

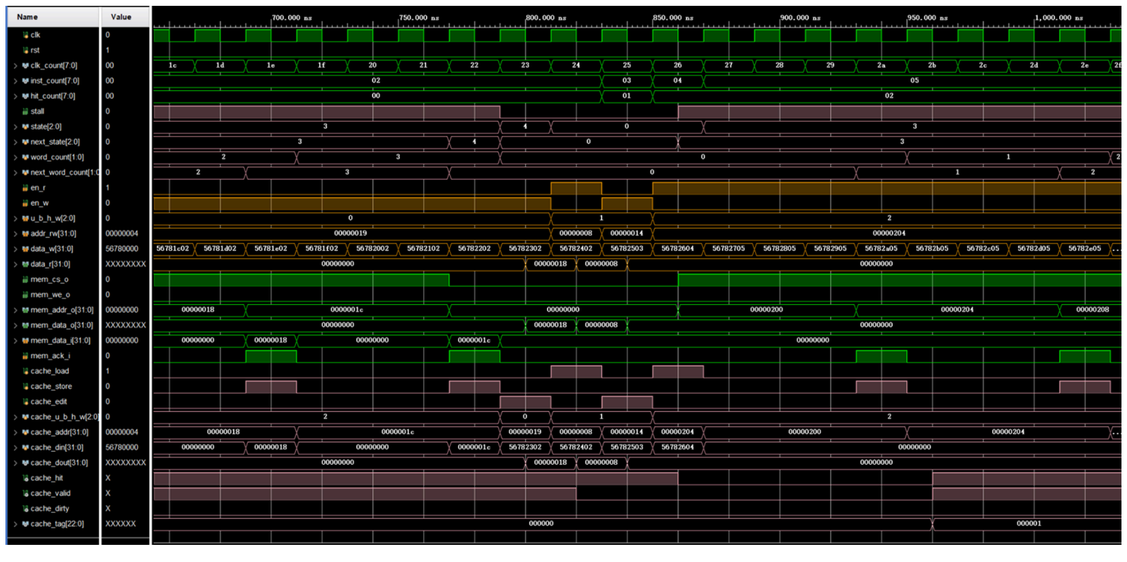
模拟 (1)



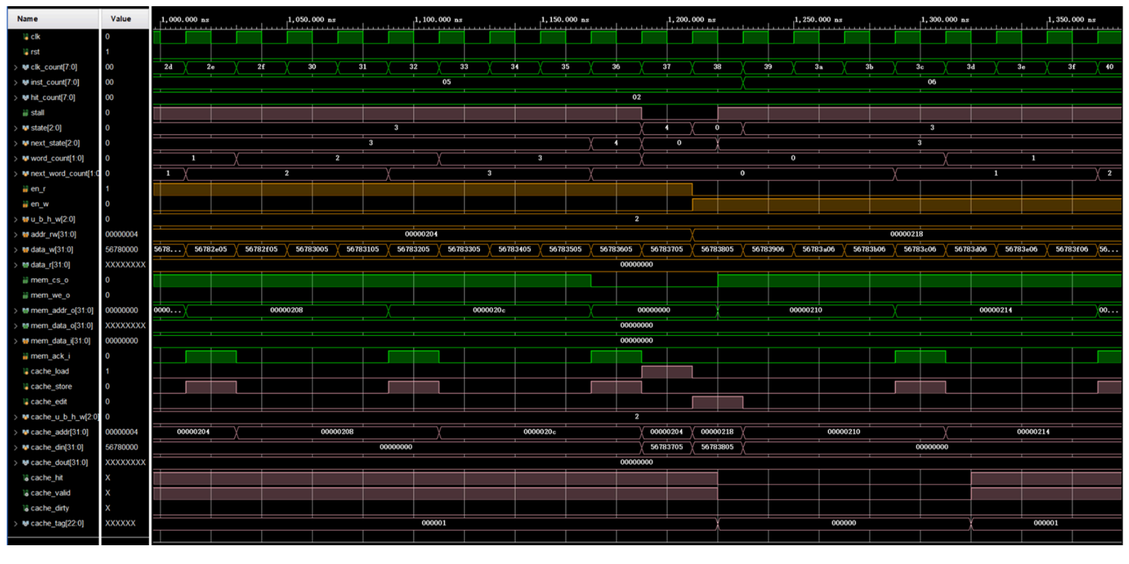
仿真 (2)



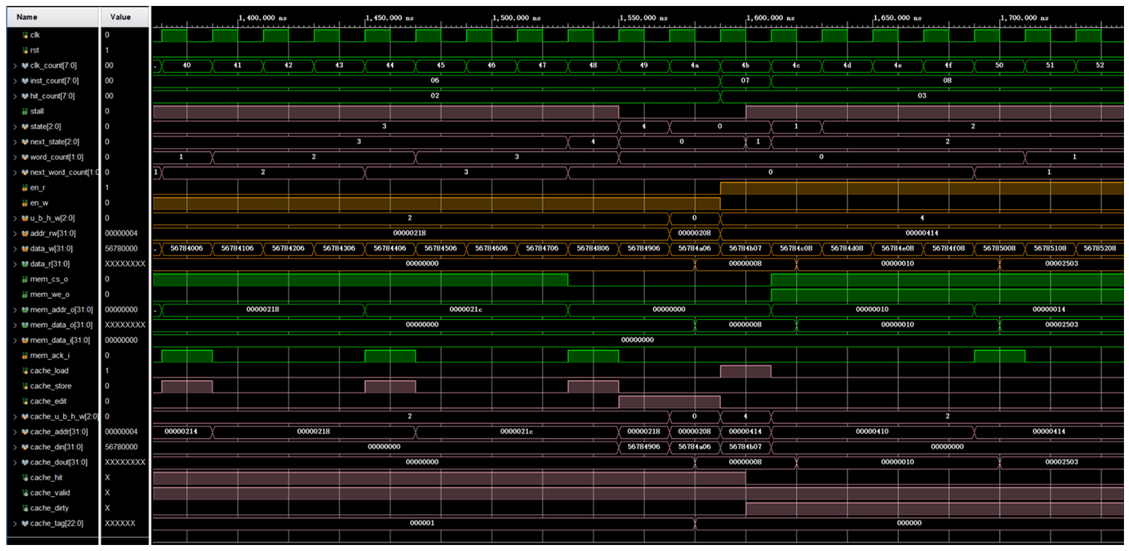
模拟（3）



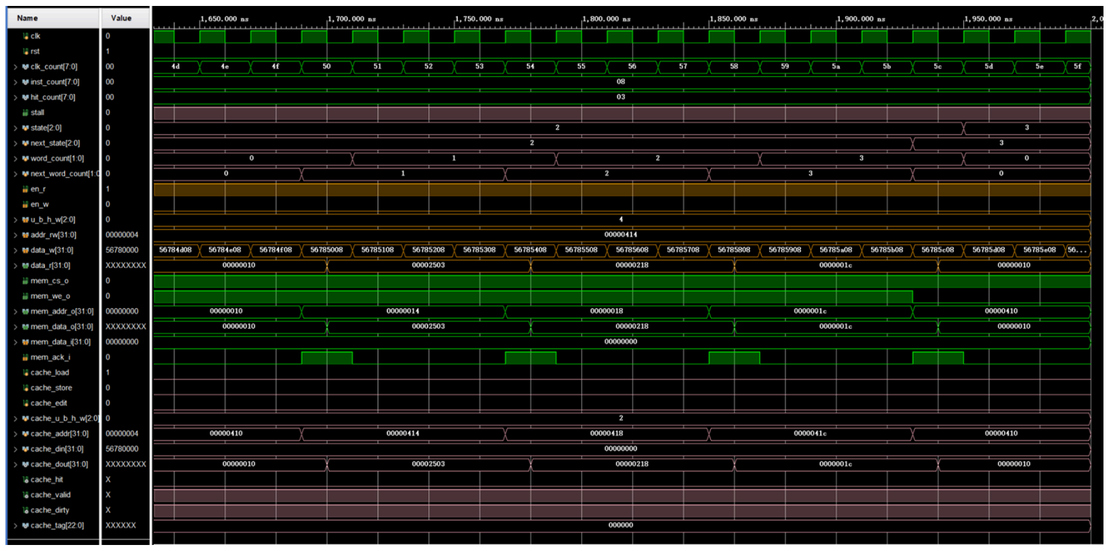
模拟（4）



模拟（5）



模拟（6）



参考文献

- 超标量处理器设计
- 现代处理器设计