浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：官泽隆

学　　院：计算机科学与技术学院

系：

专　　业：计算机科学与技术

学　　号：3180103008

指导教师：翁恺

2020 年 12 月 20 日

浙江大学实验报告

课程名称：计算机体系结构 实验类型：综合

实验项目**PCPU with cache**

学生姓名：官泽隆 专业：计算机科学与技术 学号：3180103008

同组学生姓名：/ 指导老师：翁恺

实验地点：曹光彪二期-301 实验日期：2020 年 12 月 20 日

一、 实验目的和要求

Objective:

1. 好好看！好好学！好好肝！

Task:

2. 可体会原理部分的大小标题

二、实验内容和原理

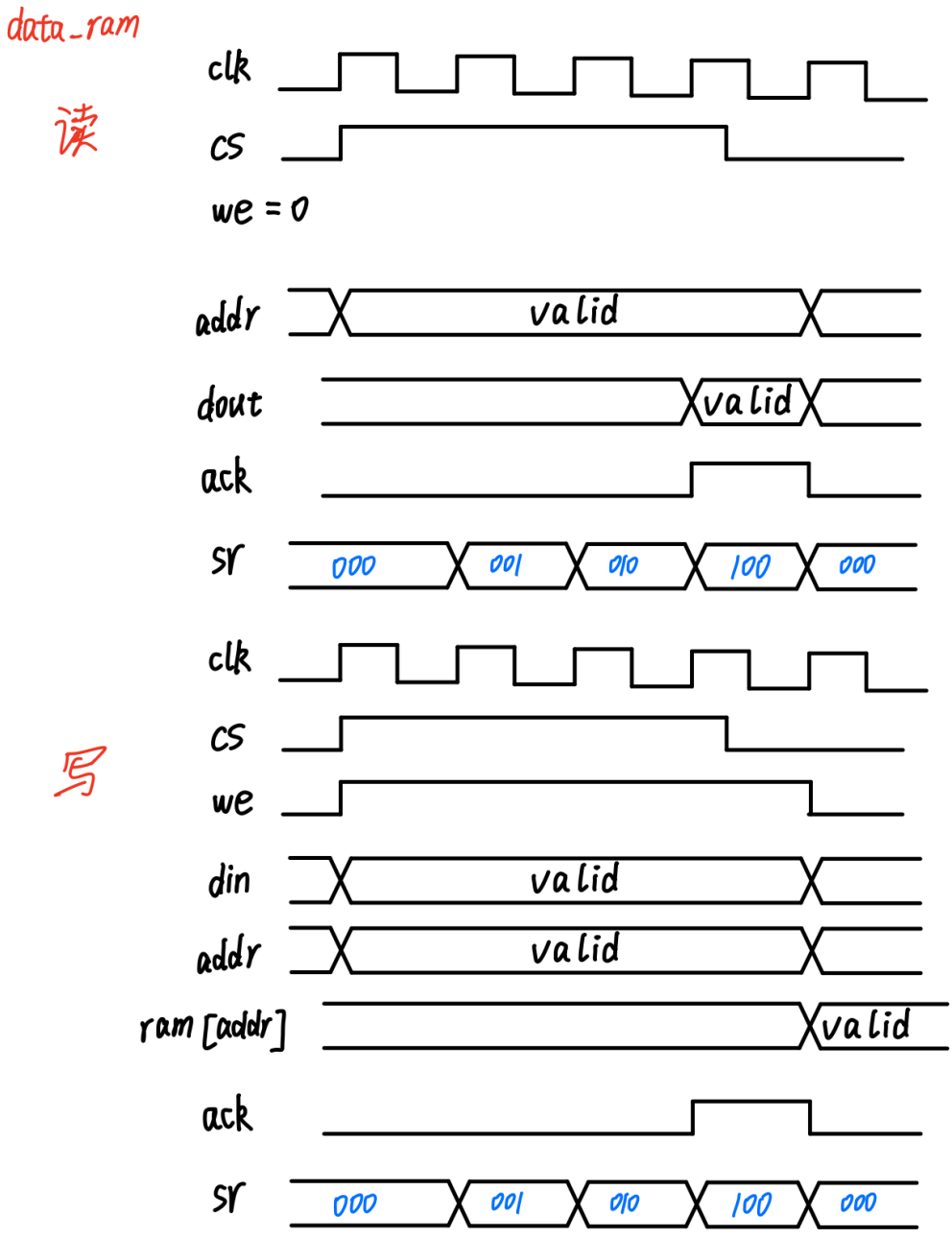
原理：

Part 1 CPU accessing Memory in multiple cycles

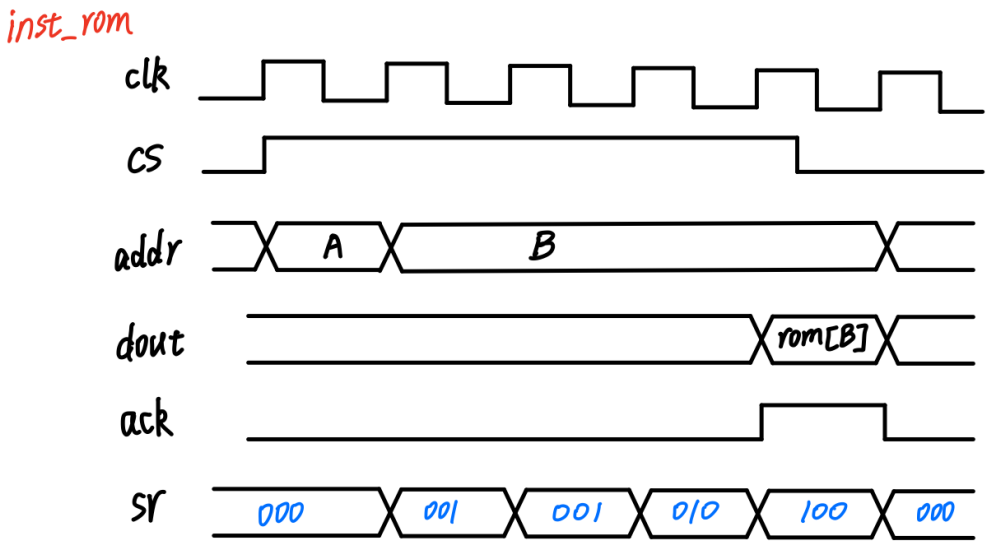
Memory 设计

利用参数化的移位寄存器提供延时和状态。

以 data_ram 为例;



注意， data_ram 对读写过程中 addr, din 等输入信号的突变是不 robust 的。但这种情况在 stall 后的 MemStage 不会出现



inst_rom 可视为关闭写入功能的 ram. 但是， inst_rom 稍微增强了对地址变化的 robustness. 采取 register 读到的 addr 值的办法；如果地址发生变化，会重置延时计数器，并阻止 ack. 这有助于在 predict not-taken 的 control hazard 应对策略下降低 stall 惩罚，因为可以藉由更改PC，提前终止错误的取指，并以正确的地址重启取指。

更加综合的流水级互锁控制

现在，stall可能发生在 IF ID MEM 级，需要理清分别的控制。

当前 in-order execution 的语境下, PCPU 可以看作一个缓行的车道：五条指令占据五级，相当于视野中有五辆车，自 IF 向 WB 方向前进。我们称 WB 方向为 “前方”；

任意一级发生 stall，相当于这辆车 broke down 了；发生 stall 这一级的指令 和 后方的指令 都不能 proceed；但不妨碍 stall 一级 前方的指令。

在逻辑上，应该自 WB 向 IF 依次考虑 stall. 例如，如果 MEM 级发生 stall，则不管 ID 级的竞争如何，都必须停顿。

```
if (mem_stall) begin
    wb_rst = -1;
    {if_CE, id_CE, ex_CE, mem_CE} = 0;
end
else if (id_stall) begin
    ex_rst = -1;
    {if_CE, id_CE} = 0;
end
else ...
```

将 发生 stall 这一级 前方的暂存器清零，达到插入气泡的目的。 在考虑 IF 级 stall 同时，要注意对 predict not-taken 的影响. 如果简单地把 IF 停住，ID 清零，就会丢失 ID 向 IF 发出的 BRANCH 信号，相当于永远 not-taken，CPU 的正确性有失。

```
if (id_BRANCH) begin
    id_rst = -1;
end
else if (if_stall) begin
    id_rst = -1;
```

```
    if_CE = 0;
end
```

因此，我让 id_BRANCH 时的行为和前一次实验一样，仅仅将 ID 清零。这将导致 IF.PC 装入分支的地址，原来 predict not-taken 的指令变成 NOP. 但这对 inst_rom 提出了要求，要能够应对 addr 的中途改变，不能按原来 predict not-taken 的 PC 取出不应执行的指令。

Part 2 Cache design

CMU 面向 CPU 和 Memory 的接口都是如下形式：

```
    input wire cs,
    input wire wea,
    input wire [ADDR_W-1:0] addra,
    input wire [31:0] dina,
    output reg [31:0] douta,
    output reg ack,
```

cache 的 主要特性：direct-mapped; write-back; write-allocate. 简单起见， 一个 block 的 data 宽和内存一样，32位。

设计 3 个状态:

```
parameter
    IDLE = 2'd0,
    BACK = 2'd1,
    ALLOCATE = 2'd2;
reg [1:0] state = IDLE;
```

IDLE 状态，检查来自 CPU 的读写请求。 如果有请求，如果地址命中，则和之前实验一样，操作在本时钟周期就能完成； 否则，需要把 cache block 读进来。 如果该块已在使用且还是脏的，次态转到写回； 否则次态转到 ALLOCATE.

BACK 状态， CMU 向 Memory 发出写请求，使用需要写回的 cache block 的地址 和 数据。同时等待 Memory 的 ack 信号，以转到 ALLOCATE 次态。

ALLOCATE 状态， CMU 向 Memory 发出读请求，将当前 CPU 地址线上的值 传给 Memory. 同时等待 Memory 的 ack 信号，以将 Memory 数据线上的内容存入 cache line，并转到 IDLE 次态。（下一周期就将命中）

三、实验过程和数据记录

1. 行为仿真——助教给的测试程序

对应 sim_mips.v

```
1   begin:
2   Lw $1, 20($zero)        //R1=4
3   Lw $2, 24($zero)        //R2=1
4   Add $3,$2,$1        //R3=5  //2LW-ALU:forwarding:1 stall
5   Sub $4,$3,$1        //R4=1  //2ALU-ALU
6   And $5,$3,$1        //R5=4  //无冲突
7   Or $6,$3,$1     //R6=5  //无冲突
8   addi $6,$3,4        //$6=9  //无冲突
9   Add $7, $zero, $1       //R7=4  //无冲突
10  Lw $8,0($7)         //R8=8  //2ALU-LW
11  Sw $8,8($7)         //  //2LW-SW: forwarding可以解决
12  Lw $9, 8($7)        //R9=8
13  Sw $7, 0($9)        //  //2LW-SW: forwarding:1 stall
14  Lw $10,0($9)        //R10=4
15  Add $11, $1, $1     //R11=8
16  Add $10, $1, $10    //R10=8 //1LW-ALU:forwarding可以解决
17  Add $10,$1,$2       //R10=5
18  Beq $10, $11 ,lable1    // not taken//2+1ALU-BEQ//  branch x
19  Lw $1, 8($7)        //R1=8
20  Lw $2, 24($zero)        //R2=1
```

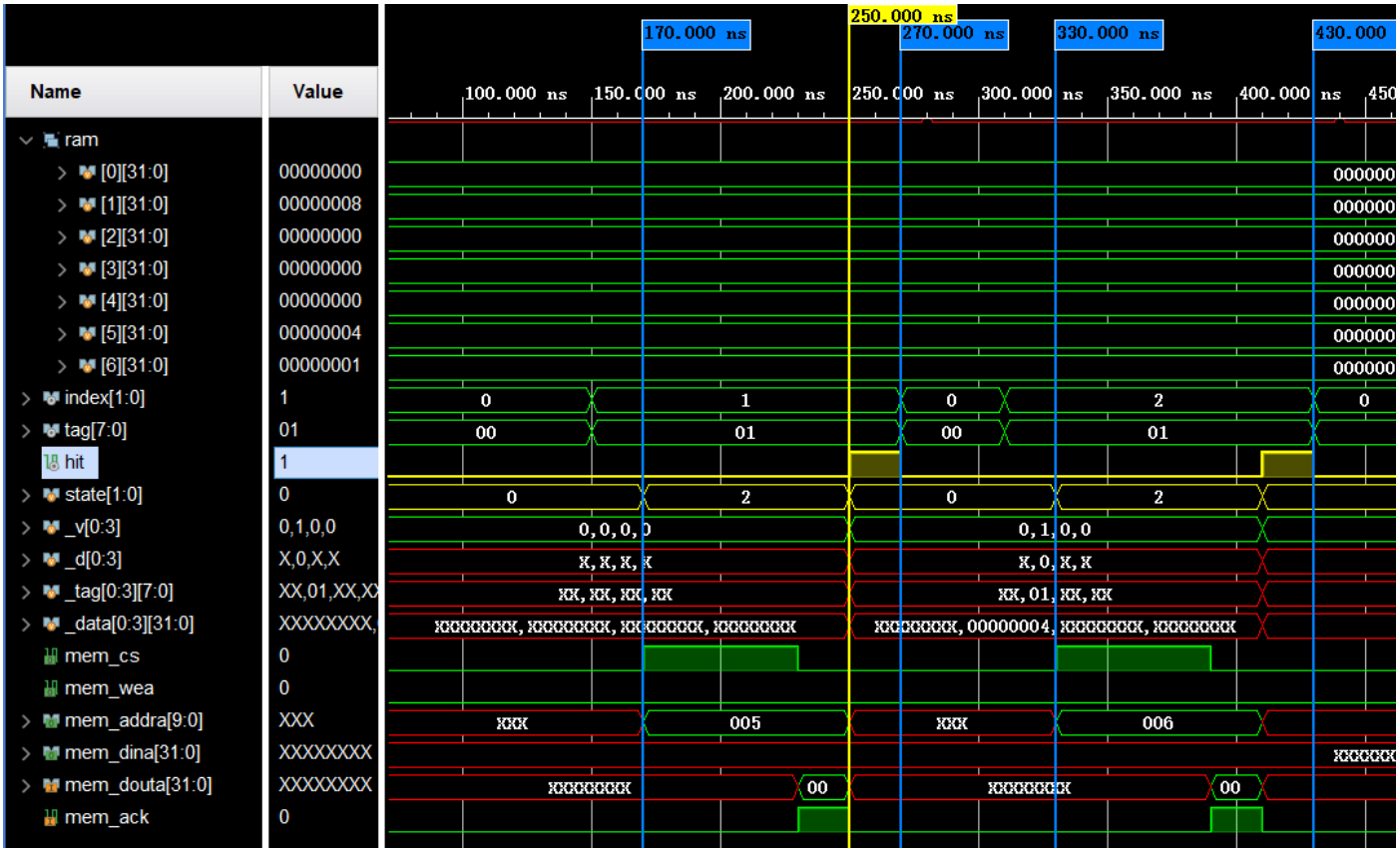
内存查看 (ROM)

00000000 跳转 上一页 下一页 查看PC指针 十

地址	数据	汇编指令	ASCII码
00000000	8C 01 00 14	LW $at, 20($zero)	----
00000004	8C 02 00 18	LW $v0, 24($zero)	----
00000008	00 41 18 20	ADD $v1, $v0, $at	-A-
0000000C	00 61 20 22	SUB $a0, $v1, $at	-a "
00000010	00 61 28 24	AND $a1, $v1, $at	-a($
00000014	00 61 30 25	OR $a2, $v1, $at	-a0%
00000018	20 66 00 04	ADDI $a2, $v1, 4	f--
0000001C	00 01 38 20	ADD $a3, $zero, $at	--8
00000020	8C E8 00 00	LW $t0, 0($a3)	----
00000024	AC E8 00 08	SW $t0, 8($a3)	----
00000028	8C E9 00 08	LW $t1, 8($a3)	----
0000002C	AD 27 00 00	SW $a3, 0($t1)	-'--
00000030	8D 2A 00 00	LW $t2, 0($t1)	-*--
00000034	00 21 58 20	ADD $t3, $at, $at	-!X
00000038	00 2A 50 20	ADD $t2, $at, $t2	-*P
0000003C	00 22 50 20	ADD $t2, $at, $v0	-"P
00000040	11 4B 00 02	BEQ $t2, $t3, 2	-K--
00000044	8C E1 00 08	LW $at, 8($a3)	----
00000048	8C 02 00 18	LW $v0, 24($zero)	----

clock cycle 20 ns

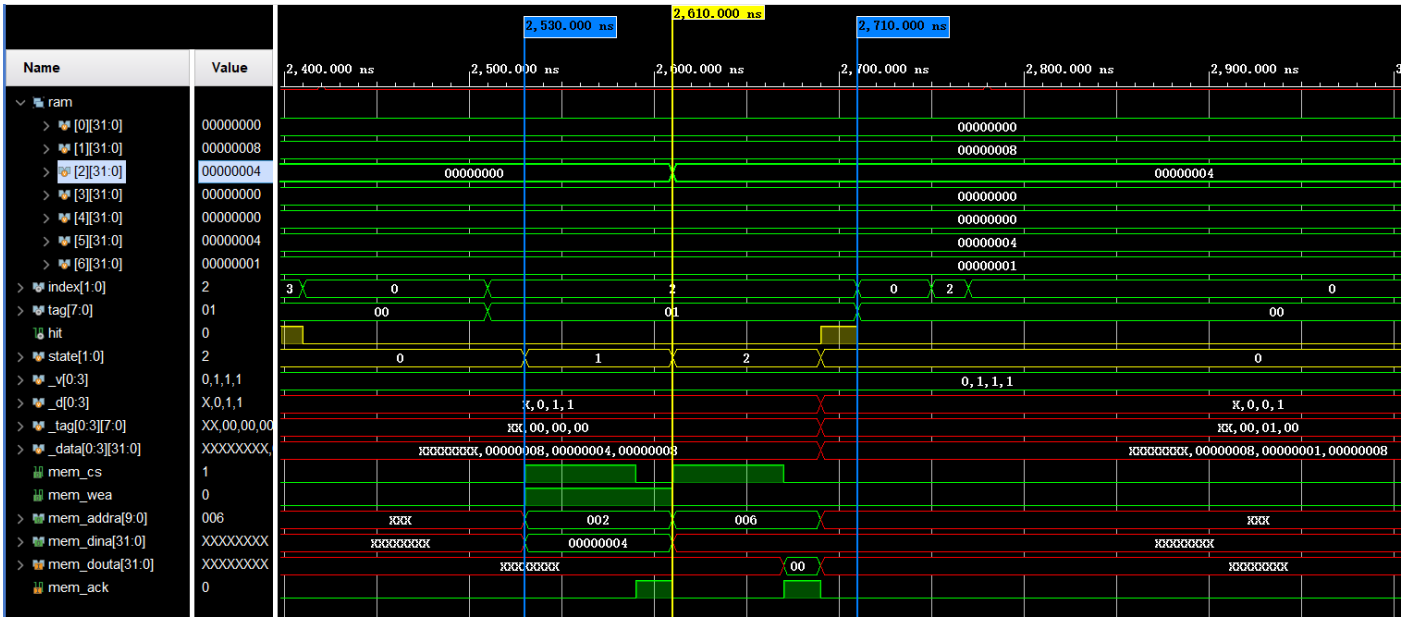
```
0x0 Lw $1, 20($zero)
```



cold-start miss；以及 ALLOCATE 后的 cache hit

----------second case----------

```
0x48 Lw $2, 24($zero)
```



cache-miss 且在 write-ALLOCATE 前引起 write-BACK