中国科学院大学计算机组成原理实验课

实 验 报 告

学号： 2019K8009915025 姓名：马月骁 专业： 计算机科学与技术

实验序号：5 实验名称： 高速缓存（cache）设计

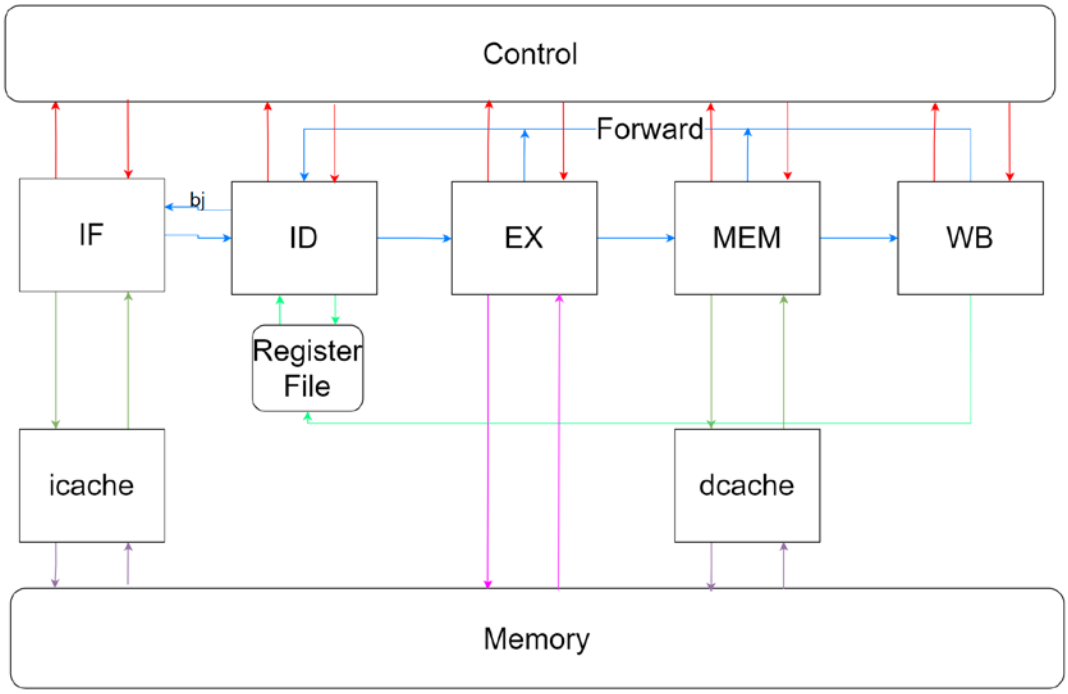
注 1：撰写此 Word 格式实验报告后以 PDF 格式保存在本地仓库的主目录下。文件命名规则：学号-prjN.pdf，其中学号中的字母“K”为大写，“-”为英文连字符，“prj”和后缀名“pdf”为小写，“N”为 1 至 4 的阿拉伯数字。例如：2019K8009929000-prj1.pdf。PDF 文件大小应控制在 5MB 以内。此外，实验项目 5 包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：学号-prj5-projectname.pdf，例如：2019K8009929000-prj5-dma.pdf。具体要求详见实验项目 5 讲义。 注 2：使用 git add 及 git commit 命令将实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 GitLab 远程仓库 master 分支（具体命令详见实验报告）。 注 3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

一、 逻辑电路结构与 FPGA 运行结果的截图及说明

高速缓存（cache）设计实验中使用 MIPS 多周期处理器以及 RSICV 五级流水线处理器。首先在 MIPS 多周期处理器上完成 icache（指令 cache）与 dcache（数据 cache）的初步设计，随后完善设计并应用在修改后的 RSICV 五级流水线处理器上。

高速缓存 (cache) 设计实验主要完成 icache.v 与 dcache.v 两个文件的设计。其余文件 custom_cpu.v；IF_stage.v，ID_stage.v，EX_stage.v，MEM_stage.v，WB_stage.v，reg_file.v，alu.v，shifter.v；my_cpu.h 在做必要修改后直接复用 prj3 以及 prj5.1（复杂处理器设计实验）中的对应文件。

以下为高速缓存（cache）设计实验中设计的基于 RSICV 五级流水线处理器的 cache 整体逻辑电路结构图：



（一）高速缓存（cache）设计以及相关代码

本次实验中 icache 以及 dcache 均使用“多周期”（状态机），并未实现流水架构。

1．icache 设计以及相关代码

（1）icache 状态机设计以及相关代码

icache 状态机设计见实验讲义，报告中不重复叙述。状态机代码如下：

```
94      //current_state
95      always @(posedge clk) begin
96          if(rst) begin
97              current_state <= WAIT;
98          end
99          else begin
100             current_state <= next_state;
101         end
102     end
```

```
    //next_state
    always @(*) begin
        case(current_state)
            WAIT: begin
                if(from_cpu_inst_req_valid) begin
                     next_state = TAG_RD;
                end
                else begin
                     next_state = WAIT;
                end
            end

            TAG_RD: begin
                if(hit_miss) begin
                    next_state = CACHE_RD;
                end
                else begin
                    next_state = EVICT;
                end
            end

            CACHE_RD: begin
                next_state = RESP;
            end

            RESP: begin
                if(from_cpu_cache_rsp_ready) begin
                    next_state = WAIT;
                end
                else begin
                    next_state = RESP;
                end
            end

            EVICT: begin
                next_state = MEM_RD;
            end

            MEM_RD: begin
                if(from_mem_rd_req_ready) begin
                    next_state = RECV;
                end
                else begin
                    next_state = MEM_RD;
                end
            end

            RECV: begin
                if(from_mem_rd_rsp_valid & from_mem_rd_rsp_last) begin
                    next_state = REFILL;
                end
                else begin
                    next_state = RECV;
                end
            end

            REFILL: begin
                next_state = RESP;
            end

            default: next_state = current_state;
        endcase
    end
```

（2）icache 替换策略与替换过程

在 MIPS 多周期处理器和 RSICV 五级流水处理器中，分别采用了 FIFO 和 PLRU 两种不同的替换策略。

FIFO：通过对应每组 cache block 添加一个 2 位计数器，每当 cpu 访问 cache 没有命中时将计数器加 1，进而选择特定一路进行替换。

FIFO 替换策略相关代码如下：

```
191     always @(posedge clk) begin
192         if(rst) begin
193             FIFO[0] <= 2'b0; FIFO[1] <= 2'b0; FIFO[2] <= 2'b0; FIFO[3] <= 2'b0;
194             FIFO[4] <= 2'b0; FIFO[5] <= 2'b0; FIFO[6] <= 2'b0; FIFO[7] <= 2'b0;
195         end
196         else if(current_state == TAG_RD) begin
197             FIFO[set] <= FIFO[set] + {1'b0, ~hit_miss};
198         end
199     end
```

FIFO 替换过程：通过添加一个 3 位计数器和 8 个 32 位寄存器来依次接收 memory 传回的数据。相关代码如下：

```
213             case(from_mem_rd_rsp_len)
214                 3'b000: from_mem_rd_rsp_data0 <= from_mem_rd_rsp_data;
215                 3'b001: from_mem_rd_rsp_data1 <= from_mem_rd_rsp_data;
216                 3'b010: from_mem_rd_rsp_data2 <= from_mem_rd_rsp_data;
217                 3'b011: from_mem_rd_rsp_data3 <= from_mem_rd_rsp_data;
218                 3'b100: from_mem_rd_rsp_data4 <= from_mem_rd_rsp_data;
219                 3'b101: from_mem_rd_rsp_data5 <= from_mem_rd_rsp_data;
220                 3'b110: from_mem_rd_rsp_data6 <= from_mem_rd_rsp_data;
221                 3'b111: from_mem_rd_rsp_data7 <= from_mem_rd_rsp_data;
222             endcase

265         if(current_state == REFILL) begin
266             case(FIFO[set])
267                 2'd0: data0[set] <= {from_mem_rd_rsp_data7, from_mem_rd_rsp_data6, from_mem_rd_rsp_data5, from_mem_rd_rsp_data4,
268                                      from_mem_rd_rsp_data3, from_mem_rd_rsp_data2, from_mem_rd_rsp_data1, from_mem_rd_rsp_data0};
269                 2'd1: data1[set] <= {from_mem_rd_rsp_data7, from_mem_rd_rsp_data6, from_mem_rd_rsp_data5, from_mem_rd_rsp_data4,
270                                      from_mem_rd_rsp_data3, from_mem_rd_rsp_data2, from_mem_rd_rsp_data1, from_mem_rd_rsp_data0};
271                 2'd2: data2[set] <= {from_mem_rd_rsp_data7, from_mem_rd_rsp_data6, from_mem_rd_rsp_data5, from_mem_rd_rsp_data4,
272                                      from_mem_rd_rsp_data3, from_mem_rd_rsp_data2, from_mem_rd_rsp_data1, from_mem_rd_rsp_data0};
273                 2'd3: data3[set] <= {from_mem_rd_rsp_data7, from_mem_rd_rsp_data6, from_mem_rd_rsp_data5, from_mem_rd_rsp_data4,
274                                      from_mem_rd_rsp_data3, from_mem_rd_rsp_data2, from_mem_rd_rsp_data1, from_mem_rd_rsp_data0};
275             endcase
276         end
```

PLRU：通过对应每组 cache block 添加一个 3 位寄存器，每当命中 cache 中特定一路时，根据路的的编码更新寄存器中的值；每当没有命中 cache 时，根据寄存器中的值选定替换的 cache block。

命中 cache 更新寄存器中的值的映射以及未命中 cache 选择更新 cache 的路数的映射如下：

表示	L2	L1	L0
Way 0	x	1	1
Way 1	x	0	1
Way 2	1	x	0
Way 3	0	x	0
替换			
Way 0	x	0	0
Way 1	x	1	0
Way 2	0	x	1
Way 3	1	x	1

注：表中 x 表示不更新

替换实际为表示（更新）的取反

PLRU 替换策略相关代码如下：

```
//PLRU
wire hit_miss_01;
assign hit_miss_01 = hit_miss_r0 | hit_miss_r1;
always @(posedge clk) begin
    if (rst) begin
        PLRU_r[0] <= 3'b0; PLRU_r[1] <= 3'b0; PLRU_r[2] <= 3'b0; PLRU_r[3] <= 3'b0;
        PLRU_r[4] <= 3'b0; PLRU_r[5] <= 3'b0; PLRU_r[6] <= 3'b0; PLRU_r[7] <= 3'b0;
    end
    else if (current_state == RESP) begin
        PLRU_r[set][0] <= hit_miss_01;
        if (hit_miss_01)
            PLRU_r[set][1] <= hit_miss_r0;
        if (~hit_miss_01)
            PLRU_r[set][2] <= hit_miss_r2;
    end
end
```

```
reg [2:0] PLRU_r [7:0];

wire valid;
wire L0;
wire L1;
wire L2;
wire L3;
assign valid = valid0[set] & valid1[set] & valid2[set] & valid3[set];
assign L0 =  valid & ~PLRU_r[set][1] & ~PLRU_r[set][0] |
            ~valid0[set];
assign L1 =  valid &  PLRU_r[set][1] & ~PLRU_r[set][0] |
            ~valid1[set] & valid0[set];
assign L2 =  valid & ~PLRU_r[set][2] &  PLRU_r[set][0] |
            ~valid2[set] & valid0[set] & valid1[set];
assign L3 =  valid &  PLRU_r[set][2] &  PLRU_r[set][0] |
            ~valid3[set] & valid0[set] & valid1[set] & valid2[set];

always @(posedge clk) begin
    if (current_state == TAG_RD) begin
        hit_miss_r  <= hit_miss;
        hit_miss_r0 <= hit_miss0;
        hit_miss_r1 <= hit_miss1;
        hit_miss_r2 <= hit_miss2;
        hit_miss_r3 <= hit_miss3;
    end
    else if (current_state == EVICT) begin
        hit_miss_r0 <= L0;
        hit_miss_r1 <= L1;
        hit_miss_r2 <= L2;
        hit_miss_r3 <= L3;
    end
end
```

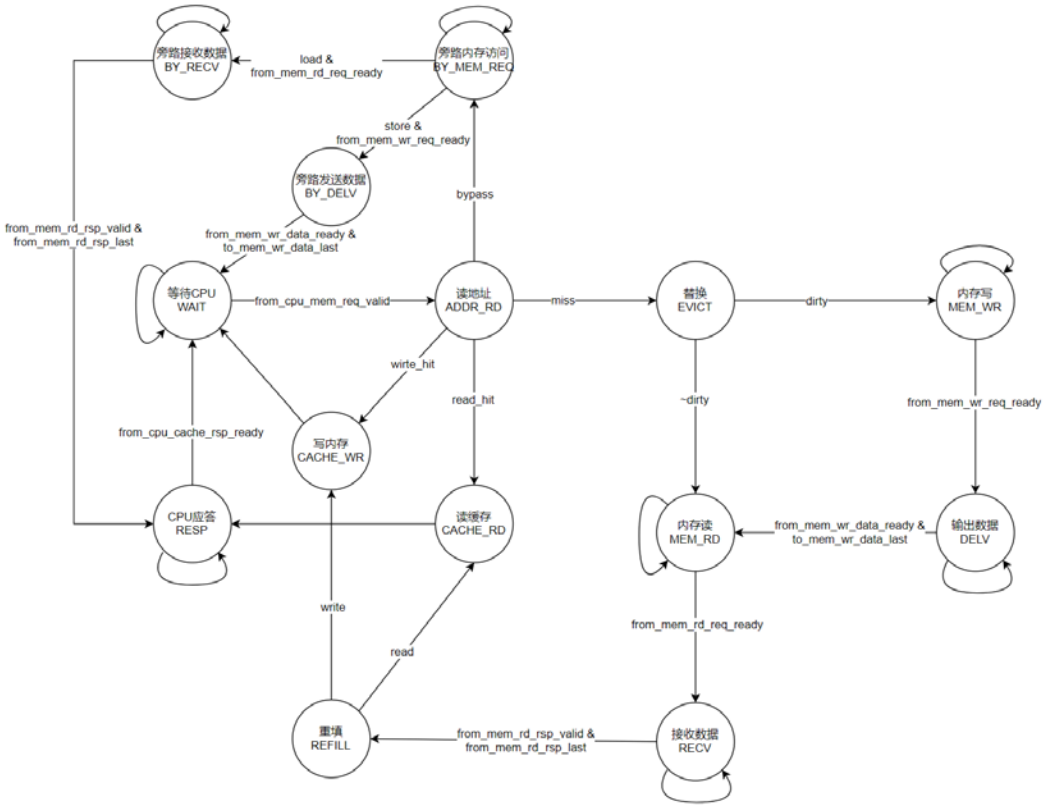
PLRU 替换过程：通过添加一个 256 位寄存器通过循环移位接收 memory 传回的数据。相关代码如下：

```
always @(posedge clk) begin
    if (current_state == RECV && from_mem_rd_rsp_valid)
        from_mem_rsp_data <= {from_mem_rd_rsp_data, from_mem_rsp_data[255:32]};
end
```

2. dcache 设计以及相关代码

（1）dcache 状态机设计以及相关代码

本次实验中 dcache 共设计了 14 个状态:



其中 BY_MEM_REQ,BY_DELV,BY_RECV 三个状态合并至 MEM_WR，MEM_RD，RECV，DELV 四个状态中，进而修改为 11 个状态的状态机。

dcache 状态机相关代码较长且与 icache 状态机相关代码类似，具体可见仓库中提交的代码，在报告中不具体列出。

（2）dcache 替换策略

dcache 中采用 PLRU 替换策略。由于 dcache 与 icache 中均采用四路组相联的设计，故 PLRU 替换的算法以及数据结构基本一致，在此不加以重复说明。

（3）dcache 与 memory 数据交互（替换过程与旁路传输）

dcache 从 memory 接收数据的相关逻辑与代码与 icache 基本保持一致，在此不加以重复叙述。

dcache 向 memory 传输数据，即突发传输（burst）：通过添加 delv_len 寄存器循环移位控制 last 信号;通过添加 delv_data 寄存器循环移位输出 32 bit 数据。

突发传输（burst）相关代码如下：

```
    //MEM_WR
    //BY_MEM_REQ
    assign to_mem_wr_req_valid = (current_state == MEM_WR) | (current_state == BY_MEM_REQ);
    assign to_mem_wr_req_addr  = {32{ bypass}}           & from_cpu_cache_addr               |
                                 {32{~bypass & h_or_m0}} & {tag0[set], set, 5'b0} |
                                 {32{~bypass & h_or_m1}} & {tag1[set], set, 5'b0} |
                                 {32{~bypass & h_or_m2}} & {tag2[set], set, 5'b0} |
                                 {32{~bypass & h_or_m3}} & {tag3[set], set, 5'b0};
    assign to_mem_wr_req_len   = {5'b0, {3{~bypass}}};

    reg [  7:0] delv_len;
    reg [255:0] delv_data;
    always @(posedge clk) begin
        if ((current_state == MEM_WR) | (current_state == BY_MEM_REQ)) begin
            if (bypass) begin
                delv_len  <= 8'b1;
                delv_data <= {224'b0, from_cpu_cache_wdata};
            end
            else begin
                delv_len  <= {1'b1, 7'b0};
                delv_data <= cache_data;
            end
        end
        else if (current_state == DELV && from_mem_wr_data_ready) begin
                delv_len  <= {1'b0, delv_len[7:1]};
                delv_data <= {32'b0, delv_data[255:32]};
        end
    end

    //DELV
    //BY_DELV
    assign to_mem_wr_data_valid = (current_state == DELV) | (current_state == BY_DELV);
    assign to_mem_wr_data_last  = delv_len[0];
    assign to_mem_wr_data_strb  = {4{~bypass}} | from_cpu_cache_wstrb;
    assign to_mem_wr_data       = delv_data[31:0];

    //MEM_RD
    //BY_MEM_REQ
    assign to_mem_rd_req_valid = (current_state == MEM_RD) | (current_state == BY_MEM_REQ);
    assign to_mem_rd_req_addr  = {32{ bypass}} & from_cpu_cache_addr
                               | {32{~bypass}} & {from_cpu_cache_addr[31:5], 5'b0};
    assign to_mem_rd_req_len   = {5'b0, {3{~bypass}}};
```

（二）FPGA 运行结果分析

基于MIPS多周期处理器设计的cache由于设计上存在潜在缺陷以及平台架构问题在通过 basic 测试后中止。

基于 RSICV 五级流水处理器设计的 cache 对基于 MIPS 多周期处理器设计的 cache 进行了一定优化，可正常通过 FPGA 上版测试：

basic (49 pass 1 / 1)，medium (150 pass 12 / 12)，advanced (184 pass 17 / 17) 三组测试正常通过

hello 打印结果如下：

```
testing 1 2 0000003
faster and "cheaper"
deadf00d % DEADF00D
00000000100000000200000000300000000400000000 5
```

以下进行 RISCV 五级流水线（cache）处理器与 RISCV 五级流水线处理器以及 RISCV 多周期处理器性能比较，由于 testbench 基本一致，故在指令一致的基础上进行两处理器的 CPI 比较。

testbench	指令总数	周期数（多周期）	周期数（五级流水）	周期数（流水cache）	CPI（多周期）	CPI（五级流水）	CPI（流水cache）
32_15pz	5224573	570658298	383339212	24136267	109.23	73.37	4.62
33_bf	452946	42701054	x	1897571	94.27	x	4.19
34_dinic	16783	1526821	1189346	143946	90.97	70.87	8.58
35_fib	2549617	198544257	199454174	10325979	77.87	78.23	4.05
36_md5	5007	403005	350087	23273	80.49	69.92	4.65
37_qsort	9572	813526	719930	38840	84.99	75.21	4.06
38_queen	81582	7077208	5736816	327112	86.75	70.32	4.01
39_sieve	10287	829272	765305	41912	80.61	74.40	4.07
40_ssort	619137	49172528	45267933	2516908	79.42	73.11	4.07

RISC-V 多周期与五级流水线处理器（cache）性能计数器数据处理表

注：五级流水处理器中 testbench 33_bf 运行出现问题，具体问题见 prj5.1 实验报告，相关数据不纳入此处结果比较。

添加 cache 的五级流水线处理器的平均 CPI 为 4.70，单独五级流水线处理器的平均 CPI 为 73.18，多周期处理器的平均 CPI 为 87.18。可以认为添加五级流水线处理器的性能相较单独五级流水线处理器和多周期处理器有大幅提升。

由上表可见,对于大多数 testbench 添加 cache 的五级流水线处理器的 CPI 稳定在 4~5，并非理想中的 1 左右。可能原因如下:

1. cpu 访问 cache 命中率有限,存在一定与内存交互的时间;若 testbench 中指令/数据分布比较分散，cache 命中率将进一步降低，导致访问内存次数增加，进而使得 CPI 上升（testbench34_dinic CPI = 8.58）
2. 本次实验中实现的 icache 与 dcache 没有使用流水线，而是多周期。即便命中 cache，cache 也需要 2~3 完成数据写入/输出。

二、 实验过程中遇到的问题、对问题的思考过程及解决方法

（一）代码逻辑错误

1. 状态机代码不规范（缺少 else 语句），综合出 latch

解决方法：根据综合报告，检查状态机，确保 if 语句后有 else 语句

2. 位宽设计错误（错将[2:0]理解为有 8bit），导致寄存器缺失

错误现象：对应位置信号始终为 x，无法写入/读出

解决方案：根据综合报告以及波形，检查对应代码

3. 将 cache 加入流水线后出现取值时序（branch）错误，导致错误取值

解决方案：根据波形，修改流水线时序（去除先前流水线设计中的 branch 指令相关冗余设计）

（二）调试过程难点

1. 开始提供的框架中 wrapper 文件存在信号命名错误以及信号连线错误

解决方案：根据波形，在确认并非自己代码错误的前提下，修改常老师提供的代码。

2. 平台不稳定，导致仿真出错或 pipeline 处于 stuck 状态

解决方案：向助教反映，及时 retry

3. 实验框架存在问题，上版出错

解决方案：检查突发传输代码，暂缓上版

三、 对讲义中思考题（如有）的理解和回答

本次实验无思考题

四、 在课后，你花费了大约 15 小时完成此次实验。

五、 对于此次实验的心得、感受和建议

（一）实验心得、感受

1. 数据通路与控制信号

在编写代码之前，先厘清所需通路与控制信号可使后续代码编写事半功倍。

2. 代码编写

在已有代码的基础上进行修改可使代码编写速度大大加快。

在修改时需要检查所有需要修改的地方，防止遗漏。如变量名修改，若存在遗漏。会导致相关信号出错。

参考以往编写代码经验，可以缩短代码编写周期。

（二）实验建议

希望早点发布。

（三）致谢

感谢楼持恒同学和我讨论 cache 相关设计问题，提供了 PLRU 替换算法的思路。