本科实验报告

课程名称：计算机体系结构

设计名称：Pipelined CPU

姓　　名：曾帅王异鸣

学　　号：3190105729 3190102780

学　　院：计算机学院

专　　业：计算机科学与技术

班　　级：图灵 1901

指导老师：姜晓红

2021 年 10 月 16 日

目录

1 实验目的

本次实验要求我们实现加入了 cache 的 CPU 流水线

1. 了解 Cache Management Unit 的运行规则以及其状态图
2. 掌握 CMU 的设计方法以及与 CPU 的交互方法
3. 掌握验证 CMU 正确性的方法，并把含有 Cache 的 CPU 与不含 Cache 的 CPU 进行比较。

2 实验内容

实验的基本要求是实现一个包含 cache 的 CPU 流水线
本实验已给出主要框架，需要完成的实验内容如下：

1. 设计 CMU 并将其整合进 CPU 中
2. 观察并分析仿真结果
3. 比较含有 Cache 与不含 Cache 的 CPU 的表现。

3 实验原理

3.1 Cache Management Unit

本实验中需要实现的主要模块就是 CMU，即 Cache Management Unit，CMU 中内含 cache 模块，其负责处理数据并将其传入或传出 cache，并完成 cache 与 CPU 和 memory 的交互。

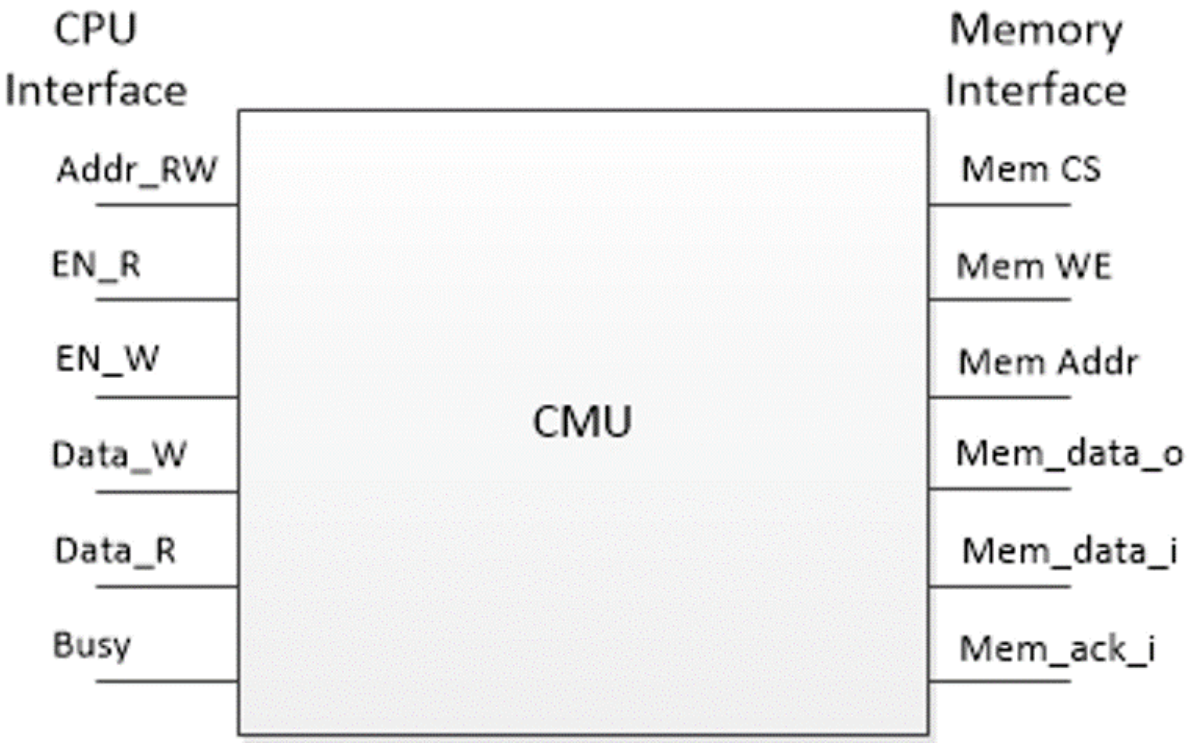

图 3.1: CMU 模块

CMU 模块安置在流水线的 MEM 阶段，其与 CPU 之间有一系列的接口，包括 Data_Read，Data_Write，Address 等，这提供了 CPU 与 cache 之间数据传输的通道。同时，CMU 还与 Memory 之间有一系列接口，以用于 cache 和 memory 之间进行数据传输。

3.2 Cache Operation Flow

CMU 中的操作流程如下图所示

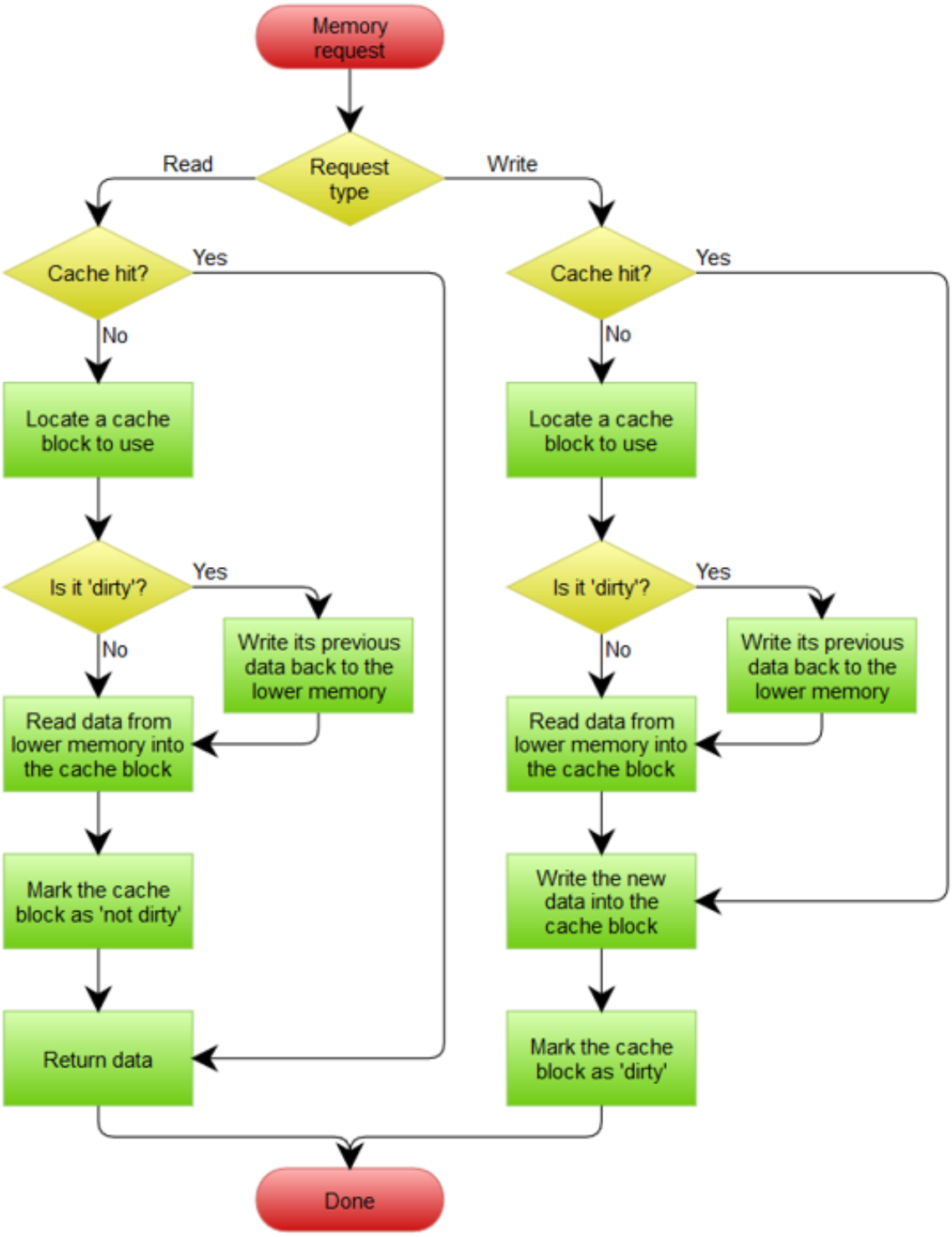

图 3.2: Cache Operation Flow

当 CPU 需要对数据进行操作时，会首先传入 read 或 write bit。CMU 接收到信号后首先查看地址是否 hit，若 hit 则直接读或写数据。否则检查需要置换的 block 是否 dirty，若 dirty 则需要先将当前块写回 memory。之后从 memory 中取数据存入 cache，

再进行读或写操作。

3.3 Cache Management State Machine

CMU 的内部结构实际上是一个有限状态机，其有 S_IDLE、S_PRE_BACK、S_BACK、S_FILL、S_WAIT 等 5 个状态。

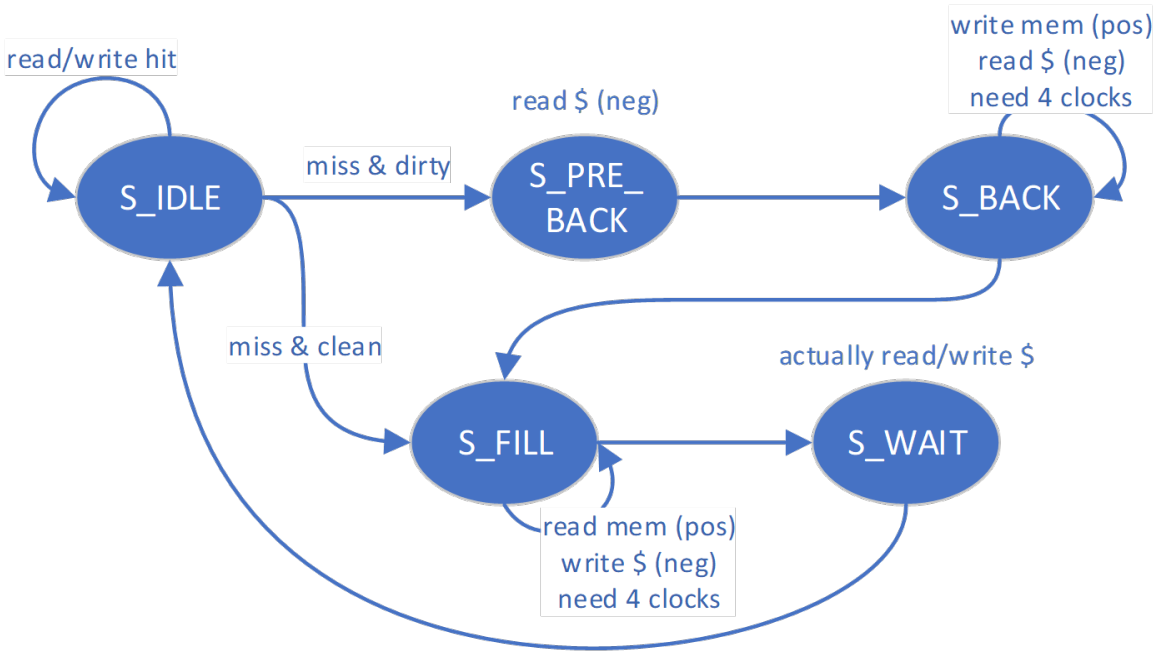

图 3.3: CMU 状态机

cache 操作均发生在当前 state 的下降沿，memory 操作均发生在当前 state 的上升沿。每个状态的具体描述如下：

1. S_IDLE：空闲状态，不进行 memory 操作，cache 操作 hit 的情况下一直处于这个状态。

2. S_PRE_BACK：为了写回，先进行一次读 cache。

3. S_BACK：上升沿将上个状态的数据写回到 memory，下降沿从 cache 读下次需要写回的数据，由计数器控制直到整个 cache line 全部写回。由于 memory 设置为 4 个周期完成读写操作，因此需要等待 memory 给出 ack 信号，才能进行状态的改变。

4. S_FILL：上升沿从 memory 读取数据，下降沿向 cache 写入数据，由计数器控制直到整个 cache line 全部写入。与 S_BACK 类似，需要等待 ack 信号。

5. S_WAIT：执行之前由于 miss 而不能进行的 cache 操作。

3.4 源代码

```
1  module cmu (
2          // CPU side
3          input clk,
4          input rst,
5          input [31:0] addr_rw,
6          input en_r,
7          input en_w,
8          input [2:0] u_b_h_w,
9          input [31:0] data_w,
10         output [31:0] data_r,
11         output stall,
12
13         // mem side
14         output reg mem_cs_o = 0,
15         output reg mem_we_o = 0,
16         output reg [31:0] mem_addr_o = 0,
17         input [31:0] mem_data_i,
18         output [31:0] mem_data_o,
19         input mem_ack_i,
20
21         // debug info
22         output [2:0] cmu_state
23         );
24
25         `include "addr_define.vh"
26
27         reg [ADDR_BITS-1:0] cache_addr = 0;
28         reg cache_load = 0;
29         reg cache_store = 0;
30         reg cache_edit = 0;
31         reg [2:0] cache_u_b_h_w = 0;
32         reg [WORD_BITS-1:0] cache_din = 0;
33         wire cache_hit;
34         wire [WORD_BITS-1:0] cache_dout;
35         wire cache_valid;
36         wire cache_dirty;
37         wire [TAG_BITS-1:0] cache_tag;
38
39         cache CACHE (
40         .clk(~clk),
41         .rst(rst),
```

```
42          .addr(cache_addr),
43          .load(cache_load),
44          .store(cache_store),
45          .edit(cache_edit),
46          .invalid(1'b0),
47          .u_b_h_w(cache_u_b_h_w),
48          .din(cache_din),
49          .hit(cache_hit),
50          .dout(cache_dout),
51          .valid(cache_valid),
52          .dirty(cache_dirty),
53          .tag(cache_tag)
54          );
55
56          localparam
57          S_IDLE = 0,
58          S_PRE_BACK = 1,
59          S_BACK = 2,
60          S_FILL = 3,
61          S_WAIT = 4;
62
63          reg [2:0]state = 0;
64          reg [2:0]next_state = 0;
65          reg [ELEMENT_WORDS_WIDTH-1:0]word_count = 0;
66          reg [ELEMENT_WORDS_WIDTH-1:0]next_word_count = 0;
67          assign cmu_state = state;
68
69          always @ (posedge clk) begin
70          if (rst) begin
71          state <= S_IDLE;
72          word_count <= 2'b00;
73          end
74          else begin
75          state <= next_state;
76          word_count <= next_word_count;
77          end
78          end
79
80          // state ctrl
81          always @ (*) begin
82          if (rst) begin
83          next_state = S_IDLE;
```

```
84          next_word_count = 2'b00;
85          end
86          else begin
87          case (state)
88          S_IDLE: begin
89          if (en_r || en_w) begin
90          if (cache_hit)
91          next_state = S_IDLE;
92          else if (cache_valid && cache_dirty)
93          next_state = S_PRE_BACK;
94          else
95          next_state = S_FILL;
96          end
97          next_word_count = 2'b00;
98          end
99
100         S_PRE_BACK: begin
101         next_state = S_BACK;
102         next_word_count = 2'b00;
103         end
104
105         S_BACK: begin //?
106         if (mem_ack_i && word_count == {ELEMENT_WORDS_WIDTH{1'b1}})     //
                2'b11 in default case
107         next_state = S_FILL;
108         else
109         next_state = S_BACK;
110
111         if (mem_ack_i)
112         next_word_count = word_count + 2'b01; //?
113         else
114         next_word_count = word_count;
115         end
116
117         S_FILL: begin
118         if (mem_ack_i && word_count == {ELEMENT_WORDS_WIDTH{1'b1}})
119         next_state = S_WAIT;
120         else
121         next_state = S_FILL;
122
123         if (mem_ack_i)
124         next_word_count = word_count + 2'b01;
```

```
125             else
126             next_word_count = word_count;
127             end
128
129             S_WAIT: begin
130             next_state = S_IDLE;
131             next_word_count = 2'b00;
132             end
133             endcase
134             end
135             end
136
137             // cache ctrl
138             always @ (*) begin
139             case(state)
140             S_IDLE, S_WAIT: begin
141             cache_addr = addr_rw;
142             cache_load = en_r;
143             cache_edit = en_w;
144             cache_store = 1'b0;
145             cache_u_b_h_w = u_b_h_w;
146             cache_din = data_w;
147             end
148             S_BACK, S_PRE_BACK: begin
149             cache_addr = {addr_rw[ADDR_BITS-1:BLOCK_WIDTH], next_word_count, {
                    ELEMENT_WORDS_WIDTH{1'b0}}};
150             cache_load = 1'b0;
151             cache_edit = 1'b0;
152             cache_store = 1'b0;
153             cache_u_b_h_w = 3'b010;
154             cache_din = 32'b0;
155             end
156             S_FILL: begin
157             cache_addr = {addr_rw[ADDR_BITS-1:BLOCK_WIDTH], word_count, {
                    ELEMENT_WORDS_WIDTH{1'b0}}};
158             cache_load = 1'b0;
159             cache_edit = 1'b0;
160             cache_store = mem_ack_i;
161             cache_u_b_h_w = 3'b010;
162             cache_din = mem_data_i;
163             end
164             endcase
```

```
165         end
166         assign data_r = cache_dout;
167
168         // mem ctrl
169         always @ (*) begin
170         case (next_state)
171         S_IDLE, S_PRE_BACK, S_WAIT: begin
172         mem_cs_o = 1'b0;
173         mem_we_o = 1'b0;
174         mem_addr_o = 32'b0;
175         end
176
177         S_BACK: begin
178         mem_cs_o = 1'b1;
179         mem_we_o = 1'b1;
180         mem_addr_o = {cache_tag, addr_rw[ADDR_BITS-TAG_BITS-1:BLOCK_WIDTH
                ], next_word_count, {ELEMENT_WORDS_WIDTH{1'b0}}};
181         end
182
183         S_FILL: begin
184         mem_cs_o = 1'b1;
185         mem_we_o = 1'b0;
186         mem_addr_o = {addr_rw[ADDR_BITS-1:BLOCK_WIDTH], next_word_count, {
                ELEMENT_WORDS_WIDTH{1'b0}}};
187         end
188         endcase
189         end
190         assign mem_data_o = cache_dout;
191
192         //important
193         assign stall = mem_ack_i | mem_cs_o;
194
195 endmodule
```

4 实验步骤与调试

4.1 仿真

根据已经写好的代码，进行仿真模拟

4.2 综合

选择左侧面板的 Run Synthesis 或者点击上方的绿色小三角，选择 Synthesis

4.3 实现

选择左侧面板的 Run Implementation 或者点击上方的绿色小三角，选择 Implementation。值得注意的是执行 implementation 之前应该确保引脚约束存在且正确，同时之前已经综合过最新的代码。

4.4 验证设计

选择左侧面板的 Open Elaborated Design，输出的结果如下，根据原理图来判断，基本没有问题

4.5 生成二进制文件

选择左侧面板的 Generate Bitstream 或者点击上上的绿色二进制标志。同时生成 Bitstream 前要确保: 之前已经综合、实现过最新的代码。如没有，直接运行会默认从综合、实现开始。此过程还要注意生成的 bit 文件默认存放在.runs 下相应的 implementation 文件夹中

4.6 烧写上板

点击左侧的 Open Hardware Manager → 点击 Open Target → Auto Connect → 点击 Program Device → 选择 bistream 路径，烧写。验证结果见实验结果部分。

5 实验结果与分析

经过综合实现，我们可以在开发板上看到测试的结果。我们分以下几个部分验证实验结果的正确性

6 讨论与心得

本实验要求在上一个实验的基础上实现中断与异常的相关功能。在本次实验中，我们再次复习了上学期与中断部分相关的内容，并学会了如何在使用流水线的情况下进行中断与异常的相关处理。在实验过程中，我们也遇到了一些问题，如中断与异常时处

理 mepc 到底有哪些区别，中断长开时应该如何处理。许多内容在课上都没有提及，我们在查阅了相关资料后得到了答案。