Report for Project 2

Bzhe's workout is none of your business

1. Members and Work divisions

(1)B06902067 (許育銘): 40 %

Wire connection in CPU.v, Dcache_top.v, Memory stall, DEBUGing.

(2)B06902053 (張維哲): 35 %

Wire connection in CPU.v, Dcache_top.v, Memory stall, Report writing, DEBUGing.

(3)B06201035 (高暐竣): 25 %

Dcache_top.v, Report writing, DEBUGing.

2. CPU.v

Add wire to **dcache_top** to control cache.

3. Control.v

```
MemtoReg_o = (Stall_i==1) ? 0 :
                ({Op_i[5:4], Op_i[0]} == 3'b001) ? 1 : 0;
MemRead_o = ({Op_i[5:4], Op_i[0]} == 3'b001) ? 1 : 0;
```

Modifying **MemtoReg_o** and adding **MemRead_o** to handle the error in the instruction only contain 0.

4. Implementation of memory stall

```
if(MemStall_i) begin
    RS1data_o <= Stall_RS1data_i;
    RS2data_o <= Stall_RS2data_i;
    imm_o <= Stall_imm_i;
    RS1_o <= Stall_RS1_i;
    RS2_o <= Stall_RS2_i;
    RD_o <= Stall_RD_i;
    MemtoReg_o <= Stall_MemtoReg_i;
    ALUCtrl_o <= Stall_ALUCtrl_i;
    MemWrite_o <= Stall_MemWrite_i;
    MemRead_o <= Stall_MemRead_i;
    ALUSrc_o <= Stall_ALUSrc_i;
    RegWrite_o <= Stall_RegWrite_i;
end
```

```
else begin
    RS1data_o <= RS1data_i;
    RS2data_o <= RS2data_i;
    imm_o <= imm_i;
    RS1_o <= RS1_i;
    RS2_o <= RS2_i;
    RD_o <= RD_i;
    MemtoReg_o <= MemtoReg_i;
    ALUCtrl_o <= ALUCtrl_i;
    MemWrite_o <= MemWrite_i;
    MemRead_o <= MemRead_i;
    ALUSrc_o <= ALUSrc_i;
    RegWrite_o <= RegWrite_i;
end
```

According to the requirement, we'll have to stall for 10 cycles if we need to access memory. In order to accomplish this, we add some wire to the pipeline register. After adding the wires, there will be two kind of input. One is the original input from **previous pipelined register** and another one is from **this pipelined register's output**. Pipelined register can choose the output by the **mem_stall**(dcache_top.p1_stall_o).

5. Dcache_Top

(1)hit

```
assign hit = (p1_tag==sram_tag) & sram_valid;
```

First, we need to check whether valid bit is 1. Then, we check whether the tag on the cache is the same as our tag.

(2)r_hit_data

```
assign r_hit_data = sram_cache_data;
```

If hit, then assign the data from cache to r_hit_data.

(3)p1_data

```
p1_data = r_hit_data[p1_offset*8 +:32];
```

Since r_hit_data is a 256 bit block of data, we only want 32 bit of it. So, we assign 32 bit of r_hit_data according to the p1_offset.

(4)w_hit_data

```
w_hit_data = sram_cache_data;
w_hit_data[p1_offset*8 +:32] = p1_data_i;
```

If hit, then assign the data from cache to w_hit_data. Then, we'll have to modify 32 bits of w_hit_data according to the p1_offset,

(5)action controller

In state controller, we modify mem_enable, mem_write, cache_we and write_back to **decide the action taken in next state.**

(a) Mem_enable

Here mem_enable represents that we're going to handle the memory. Once the data memory receive mem_enable, it'll count for ten cycles then send back ack to dcache_top.

(b) Mem_write

Data memory makes use of mem_write to distinguish whether to write the memory or read.

(c) Cache_we

Cache_we means that whether we can write cache or not.

(d) Write_back

Write_back is used for handling the correct memory to modified.

So, we can handle the cache and memory by this variable.

(6)State Controller

Here, we'll briefly introduce the **action to cache or memory taken in each state**.

(a)STATE_MISS/STATE_IDLE

```
mem_enable <= 1'b0;
mem_write <= 1'b0;
cache_we   <= 1'b0;
write_back <= 1'b0;
```

No action to cache or memory.

(b)STATE_READMISS

```
mem_enable <= 1'b1;
mem_write <= 1'b0;
cache_we <= 1'b0;
write_back <= 1'b0;
```

Read data from memory.

(c)STATE_READMISSOK

```
mem_enable <= 1'b0;
mem_write <= 1'b0;
cache_we <= 1'b1;
write_back <= 1'b0;
```

Write the data from memory to cache.

(d)STATE_WRITEBACK

```
mem_enable <= 1'b1;
mem_write <= 1'b1;
cache_we <= 1'b0;
write_back <= 1'b1;
```

Write the cache to memory.

The variable listed above is the value that needed in each stage, however, we have to change it when we're going to move to that state.

```
if(sram_dirty) begin
    // TODO: add you code here!
    mem_enable <= 1'b1;
    mem_write <= 1'b1;
    cache_we <= 1'b0;
    write_back <= 1'b1;
    state <= STATE_WRITEBACK;
end
else begin
    // TODO: add you code here!
    mem_enable <= 1'b1;
    mem_write <= 1'b0;
    cache_we <= 1'b0;
    write_back <= 1'b0;
    state <= STATE_READMISS;
end
```

If we're going to enter STATE_WRITEBACK, we'll have to set the variable into STATE_WRITEBACK type. In order to take action right at the moment it entering the new state.

6. testbench.v

We modified the output format of cycle and registers to follow the format of ref.

7. Difficulties encountered and solutions

(1) Since we use blocking assignment for all pipeline registers and signals in control, the initial assignment may conflict with the first posedge of clk. Therefore we initialize the registers after 1 sec.

(2) We misunderstand the meaning of the mem_enable, mem_write, cache_we and write_back. We list all the action taken in each state to verify the usage of those variable.