Computer Architecture Final Project Report

Name: Yu-Ting Cheng 鄭宇廷

Team: 59

ID: B08202013

1. Test Pattern Execution Cycle Number (with cache)

Test Pattern	Execution Cycle Number
I0	88
I1	444
I2	200
I3	758

2. Register Table

```
Inferred memory devices in process
        in routine CHIP line 260 in file
                '/home/raid7_2/userb08/b8202013/CA/final_project-6/final_project/01_RTL/CHIP.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|  DMEM_wdata_r_reg   | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
|  proc_finish_r_reg  | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|       PC_reg        | Flip-flop |  31   |  Y  | N  | Y  | N  | N  | N  | N  |
|       PC_reg        | Flip-flop |   1   |  N  | N  | N  | Y  | N  | N  | N  |
|   IMEM_cen_r_reg    | Flip-flop |   1   |  N  | N  | N  | Y  | N  | N  | N  |
|    finish_r_reg     | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|   DMEM_cen_r_reg    | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|   DMEM_wen_r_reg    | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|   DMEM_addr_r_reg   | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
===============================================================================
```

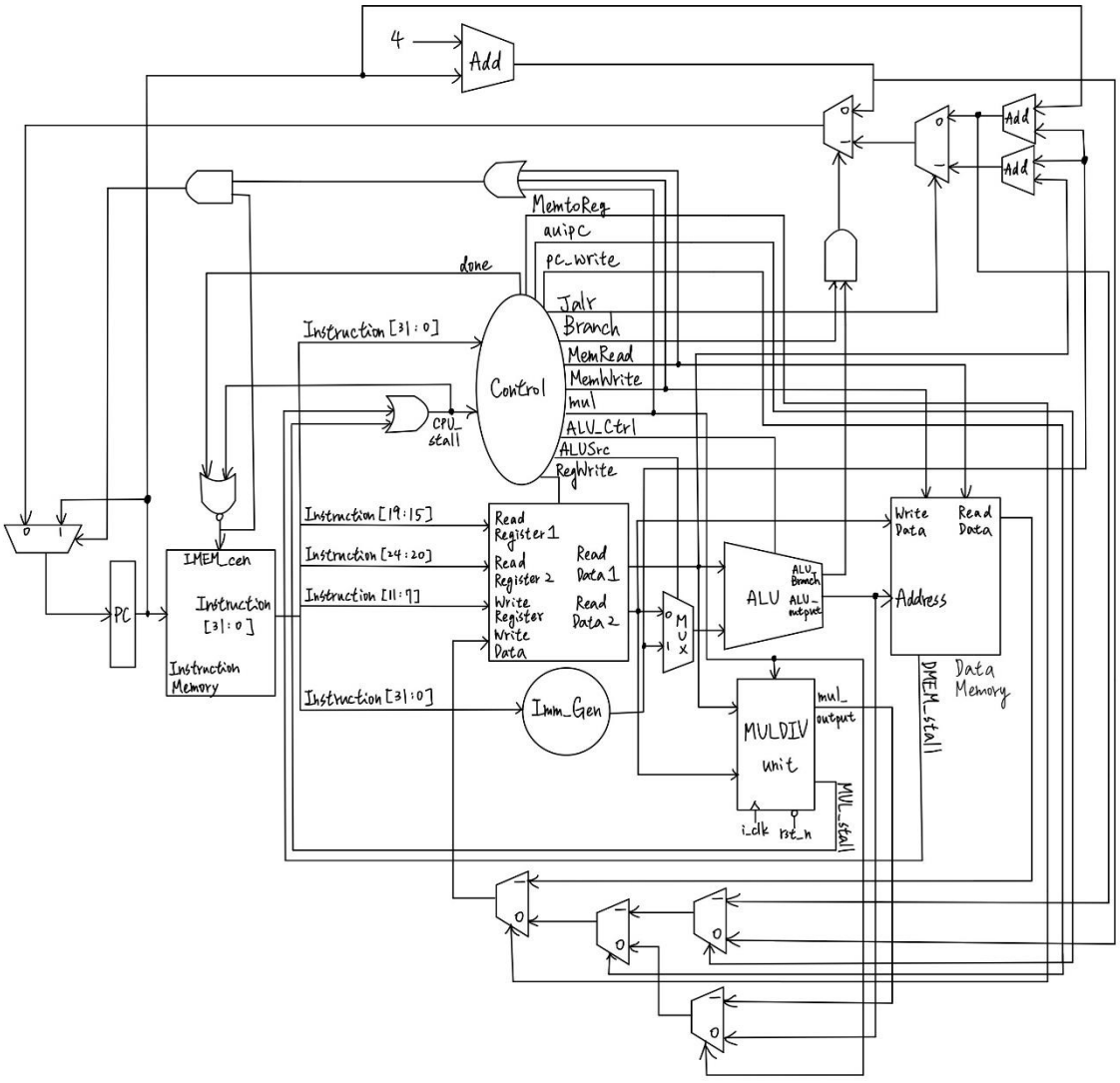
```
Inferred memory devices in process
        in routine Reg_file line 317 in file
                '/home/raid7_2/userb08/b8202013/CA/final_project-6/final_project/01_RTL/CHIP.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|       mem_reg       | Flip-flop |  995  |  Y  | N  | Y  | N  | N  | N  | N  |
|       mem_reg       | Flip-flop |  29   |  Y  | N  | N  | Y  | N  | N  | N  |
===============================================================================
```

```
Inferred memory devices in process
        in routine MULDIV_unit line 838 in file
                '/home/raid7_2/userb08/b8202013/CA/final_project-6/final_project/01_RTL/CHIP.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|     ready_r_reg     | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|     stall_r_reg     | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|     state_r_reg     | Flip-flop |   3   |  Y  | N  | Y  | N  | N  | N  | N  |
|    counter_r_reg    | Flip-flop |   9   |  Y  | N  | Y  | N  | N  | N  | N  |
|    output_r_reg     | Flip-flop |  64   |  Y  | N  | Y  | N  | N  | N  | N  |
===============================================================================
```

```
Inferred memory devices in process
        in routine Cache line 1303 in file
                '/home/raid7_2/userb08/b8202013/CA/final_project-6/final_project/01_RTL/CHIP.v'.
================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
================================================================================
|     dirty_r_reg     | Flip-flop |   32  |  Y  |  N |  Y |  N |  N |  N |  N |
|     cache_r_reg     | Flip-flop |  1024 |  Y  |  N |  Y |  N |  N |  N |  N |
|     state_r_reg     | Flip-flop |    3  |  Y  |  N |  Y |  N |  N |  N |  N |
|     write_r_reg     | Flip-flop |    1  |  N  |  N |  Y |  N |  N |  N |  N |
|    mem_cen_r_reg    | Flip-flop |    1  |  N  |  N |  Y |  N |  N |  N |  N |
|    mem_wen_r_reg    | Flip-flop |    1  |  N  |  N |  Y |  N |  N |  N |  N |
|   mem_addr_r_reg    | Flip-flop |   32  |  Y  |  N |  Y |  N |  N |  N |  N |
|   proc_addr_r_reg   | Flip-flop |   30  |  Y  |  N |  Y |  N |  N |  N |  N |
|  proc_wdata_r_reg   | Flip-flop |   32  |  Y  |  N |  Y |  N |  N |  N |  N |
|  proc_rdata_r_reg   | Flip-flop |   32  |  Y  |  N |  Y |  N |  N |  N |  N |
|   mem_wdata_r_reg   | Flip-flop |  128  |  Y  |  N |  Y |  N |  N |  N |  N |
|  proc_stall_r_reg   | Flip-flop |    1  |  N  |  N |  Y |  N |  N |  N |  N |
|   addr_real_r_reg   | Flip-flop |   32  |  Y  |  N |  Y |  N |  N |  N |  N |
|     check_r_reg     | Flip-flop |    8  |  Y  |  N |  Y |  N |  N |  N |  N |
|     store_r_reg     | Flip-flop |    1  |  N  |  N |  Y |  N |  N |  N |  N |
| cache_finish_r_reg  | Flip-flop |    1  |  N  |  N |  Y |  N |  N |  N |  N |
|  store_done_r_reg   | Flip-flop |    1  |  N  |  N |  Y |  N |  N |  N |  N |
|  set_offset_r_reg   | Flip-flop |    5  |  Y  |  N |  Y |  N |  N |  N |  N |
|    record_r_reg     | Flip-flop |    4  |  Y  |  N |  Y |  N |  N |  N |  N |
|     valid_r_reg     | Flip-flop |   32  |  Y  |  N |  Y |  N |  N |  N |  N |
|      tag_r_reg      | Flip-flop |  832  |  Y  |  N |  Y |  N |  N |  N |  N |
================================================================================
```

3. Work Description

(1) CPU Architecture Block Diagram (without cache)



(2) How you design the data path of instructions not referred in the lecture slides? (jal, jalr, auipc, …)

For those instruction, I specify some cases in *Control* module and independent control signals (e.g., o_auipc, o_jalr, etc.) such that I can detect those instructions not referred in the lecture slides. Furthermore, as you can see in the block diagram, I design the data paths of register write data and next_PC such that the CPU can perform as I expected. Those control signals help me design customized data path for each of instruction.

(3) How you handle multi-cycle instructions? (mul, div, …)

For multiplication instruction, I design finite state machine (FSM) to achieve the task. Similar to data memory stall operation, I design mul_stall and mul_ready signals such that CPU can know when to stall and when to receive the multiplication output. The FSM for multiplication is similar to HW2 design.

(4) Observation

Since the clock period is fixed and area is not considered in this project, it's feasible to list out all of instructions in different case separately. For cache part, since the bandwidth between cache and memory is 128-bits, the transferred data between cache and memory has to be consecutive 4 32-bits data. Also, due to the boundary of memory, we have to consider the offset of address to avoid accessing invalid address. Therefore, the address sent to memory is different from the address received from CPU.

4. Cache Design

(1) Cache Architecture

I implement 2-way set associative cache with write back, write allocate and LRU policy in my final cache design. There're total 16 sets (blocks) in the cache and the block size is 64-bits. The address structure is shown in the table below:

Tag	Block index	Byte offset
addr[31:6]	addr[5:2]	addr[1:0]

Since synthesizable Verilog syntax doesn't allow 3-dinemsion array, I modify the 2-way set associative design from direct-mapped cache with 32 blocks and 32-bits per block. In my final design, I still have 32 32-bits blocks, but I treats them as 2-way set associative so as to increase associativity.

Since there're two entries for each set, I use LRU (least-recently used) replacement policy to decide which set should be written back. I use a track bit for each set to track

which set is the least-recently used. Without LRU policy, if I always access the first set, generally it would perform like direct-mapped cache, which is not I desired. Finally, I use **2233 registers** to implement all of my cache design above.

(2) How your cache improves time performance?

We can examine test pattern I3 and see the time performance improvement:

Test Pattern	Without Cache	With Cache	Speedup
I3	1464	**758**	**1.93**

5. Work Distribution Table

I don't have team member. So, **100% to my final project is contributed by myself**.