Computer Architecture Final Project Report

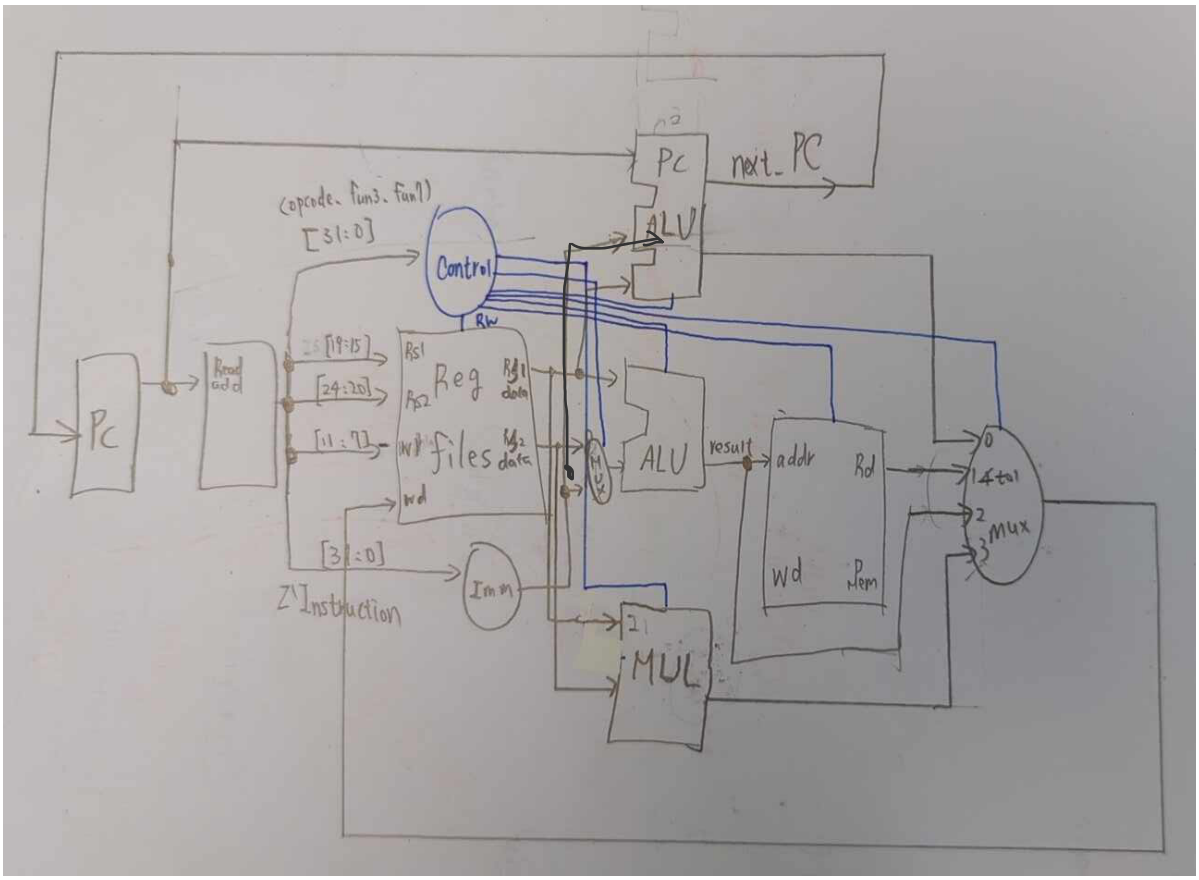
B10901091 李冠儀、B10901158 徐御宸

1. Execution time for each Instruction set:
   (the screenshot is in the last pages)

Instruction Set	Without Cache	With Cache	Speed Up
I0	78	59	1.322
I1	463	379	1.222
I2	421	375	1.123
I3	1359	595	2.284

(Unit: Total Execution Cycle)

2. The block diagram of your CPU architecture



3. Data path of instructions
   A. For ALU instruction (ex. Add, sub, xor, and, addi, slli, slti, srai), we just go through PC → Instruction memory → Reg files(Read) → ALU → Reg files (write)

B. For MUL instruction, we just go through PC → Instruction memory → Reg files(Read) → MUL → Reg files (write)

C. For pc association instruction(Auipc, Jalr, Jal, beq, bge, blt, bne), We design a PC-nxt function to deal with PC. In the picture above, as you can see, we go through PC → Instruction memory → Reg files(Read) → PCALU → Reg files (write)

*this is our PCALU

```
//pcnxt function
always @(*) begin
    if(instruction[6:0] == 7'b1101111) begin
        next_PC = PC + {{12{instruction[31]}}, instruction[19:12], instruction[20], instruction[30:21], 1'b0};
    end
    else if(instruction[6:0] == 7'b1100111) begin
        next_PC = {{20{instruction[31]}},instruction[31:20]} + reg_rdata1;
    end
    else if(instruction[6:0] == 7'b1100011) begin
        case(instruction[14:12])
            3'b000: begin
                if(reg_rdata1 == reg_rdata2)
                    next_PC = PC + {{12{instruction[31]}}, instruction[7], instruction[30:25], instruction[11:8], 1'b0};
                else
                    next_PC = PC + 32'h4;
            end
            3'b001: begin
                if(reg_rdata1 != reg_rdata2)
                    next_PC = PC + {{12{instruction[31]}}, instruction[7], instruction[30:25], instruction[11:8], 1'b0};
                else
                    next_PC = PC + 32'h4;
            end
            3'b100: begin
                if(reg_rdata1 < reg_rdata2)
                    next_PC = PC + {{12{instruction[31]}}, instruction[7], instruction[30:25], instruction[11:8], 1'b0};
                else
                    next_PC = PC + 32'h4;
            end
            3'b101: begin
                if(reg_rdata1 >= reg_rdata2)
                    next_PC = PC + {{12{instruction[31]}}, instruction[7], instruction[30:25], instruction[11:8], 1'b0};
                else
                    next_PC = PC + 32'h4;
            end
            default: begin
                next_PC = PC + 32'h4;
            end
        endcase
    end
    else if (state_nxt != S_INIT)begin
        next_PC = PC;
    end
    else begin
        next_PC = PC + 32'h4;
    end
```

D. For LW and SW instruction, we go through PC → Instruction memory → Reg files(Read) → ALU → Data memory → Reg files (write)

E. For Ecall, just set finish = 1, trun off all control signal.

4. Handle multi-cycle instructions

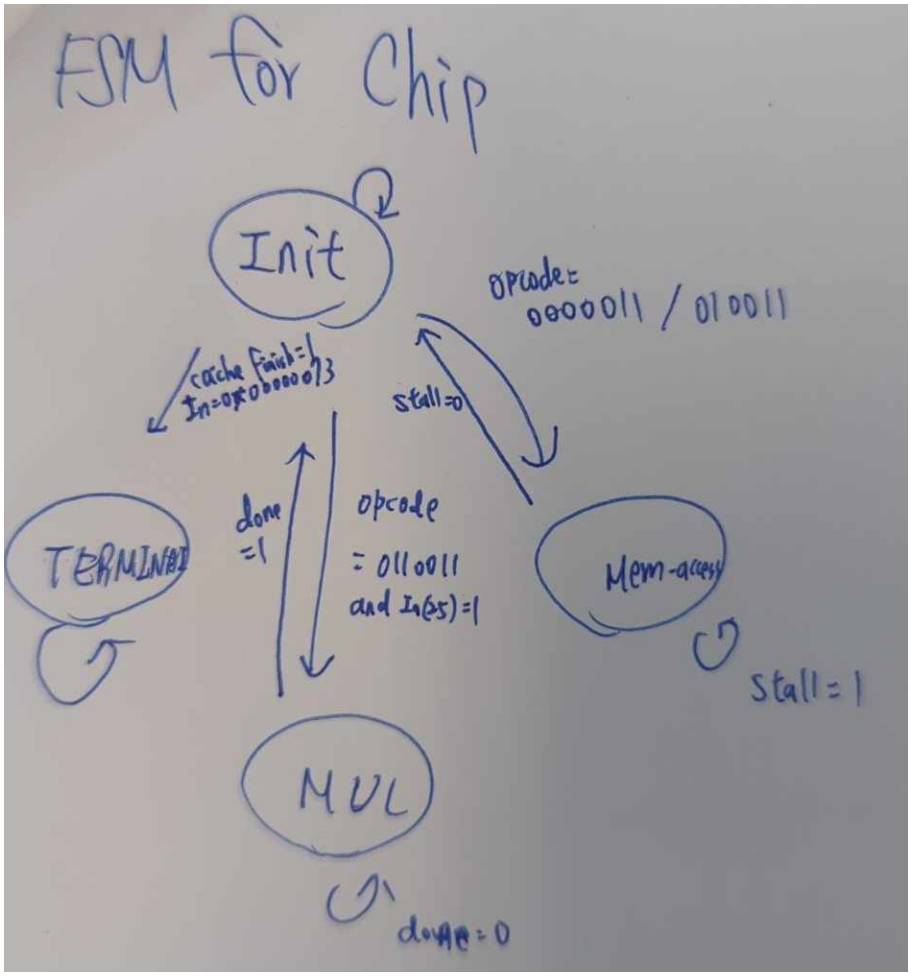
   The picture below is our FSM for chip. For the multi-cycle instructions:

   A. MUL

      When we receive MUL opcode, sent control signal and data to MULDIV_unit, the state will move to MUL state. Then, it will write result to register and go back to INIT state again until done = 1.

   B. LW and SW

      When we receive LW or SW opcode, sent control signal and data to data memory(or cache), the state will move to MEM-access state. Then, it will write result to register if need and go back to INIT state again until stall = 0.



5. Observation

   A. We have faced some problem due to immediate, because some immediate only sent less than 32 bits. So we need to do sign extension, like the picture below:

```
next_PC = PC + {{12{instruction[31]}}, instruction[7], instruction[30:25], instruction[11:8], 1'b0};
```

B. When we compare two signed data, we can't just use ">", "<"

```
3'b010: result = ( ((reg_rdata1[31] == 1'b1) && (instruction[31] == 1'b0))
    || ((reg_rdata1[31] == instruction[31]) && (reg_rdata1 < {{20{instruction[31]}},instruction[31:20]}))
    ) ? 32'b1 : 32'b0;
```

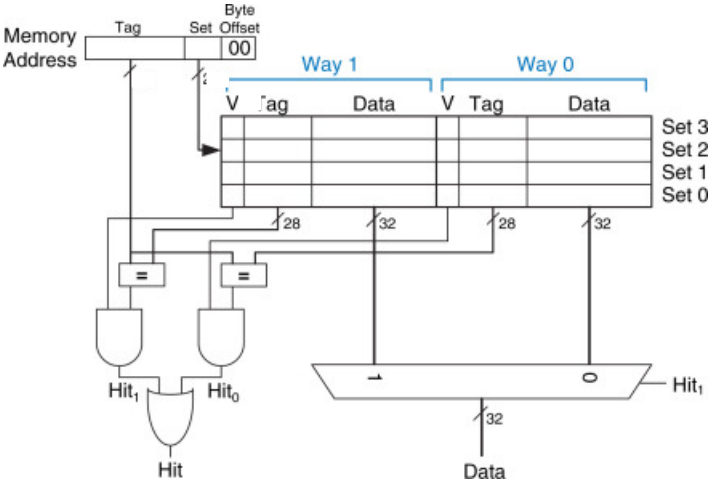
C. In instruction memory, the instruction will update with new PC when clk falling edge. Hence, for one cycle instruction, the first half cycle (clk = 1) do instruction-fetching, and the second half cycle (clk = 0) do operation. Ex. Register write operation only can be done in second half cycle.

D. Register table for chip

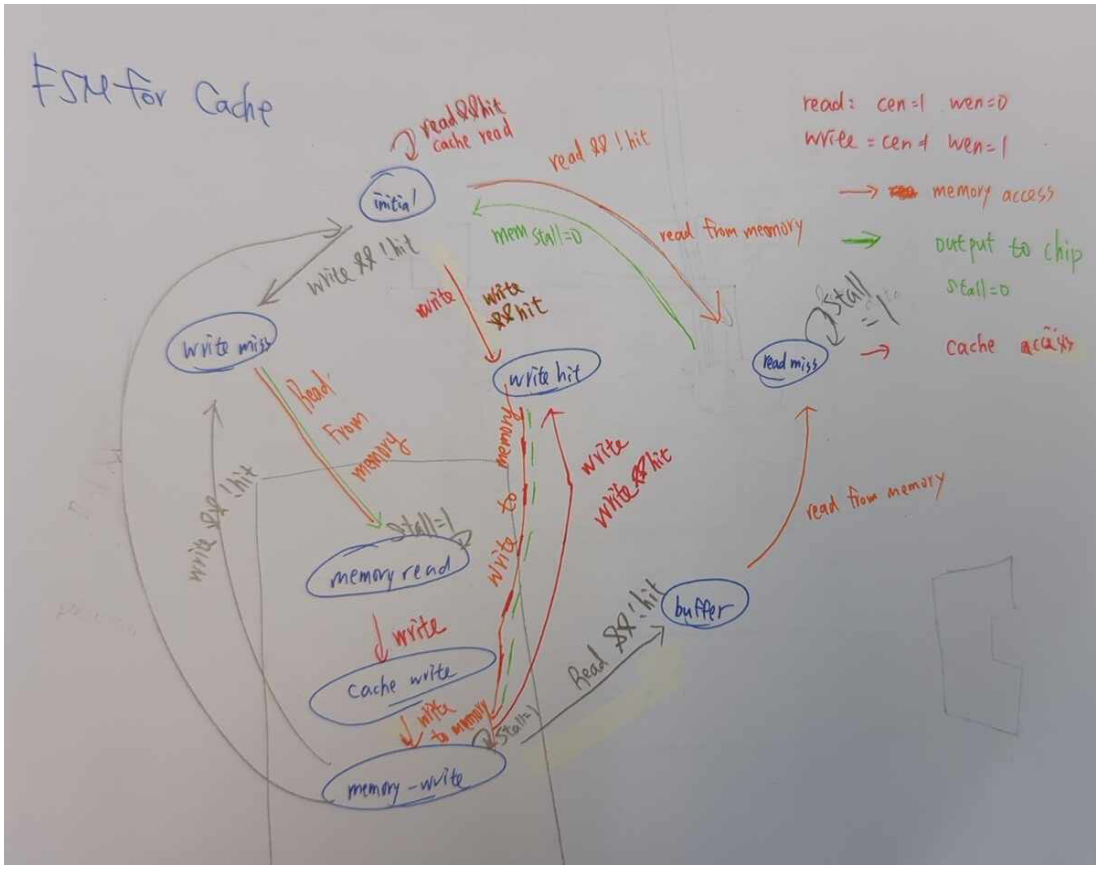
```
/home/ra10/_2/user010/b10158/that_project/01_RTL/CHIP.v :
=================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=================================================================================
|       PC_reg        | Flip-flop |  31   |  Y  | N  | Y  | N  | N  | N  | N  |
|       PC_reg        | Flip-flop |   1   |  N  | N  | N  | Y  | N  | N  | N  |
|     state_reg       | Flip-flop |   2   |  Y  | N  | Y  | N  | N  | N  | N  |
=================================================================================
```

Cache

6. Cache Architecture
   A. We use L1 2-way cache (implement with write through).
      And 1 data is 128 bit, data replace by least recently used
      Address = Real Address - offset
      Tag = address[31:7], Index = address[6:4],
      block offset = address[3:2], Byte offset = address [1:0]



B. We use 8 state FSM



A. When receiving write command, we will set stall = 0, and make chip can do the next instruction. Meanwhile, cache can access memory.

B. If cache still do the command, and cache receive new command needed memory access, we will set stall = 1 until present command is finished, and then do the new command.

7. Performance Improve

Cache can reduce time to access memory. Ex. Read hit only need 2 clocks. Specially, I3 speed up 2.284x because it has a lot of LW and SW.

Instruction Set	Without Cache	With Cache	Speed Up
I0	78	59	1.322
I1	463	379	1.222
I2	421	375	1.123
I3	1359	595	2.284

(Unit: Total Execution Cycle)

E.Register table for cache

```
================================================

Inferred memory devices in process
        in routine Cache line 1020 in file
                '/home/raid7_2/userb10/b10158/final_project/01_RTL/CHIP.v'.
=============================================================================================
|     Register Name     |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=============================================================================================
|    chip_wdata_reg     | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
| memory_write_addr_reg | Flip-flop |  30   |  Y  | N  | Y  | N  | N  | N  | N  |
| memory_write_data_reg | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
|       state_reg       | Flip-flop |   4   |  Y  | N  | Y  | N  | N  | N  | N  |
|      tag_lru_reg      | Flip-flop |   8   |  Y  | N  | Y  | N  | N  | N  | N  |
|       mem1_reg        | Flip-flop | 1024  |  Y  | N  | Y  | N  | N  | N  | N  |
|       tag2_reg        | Flip-flop |  200  |  Y  | N  | Y  | N  | N  | N  | N  |
|       mem2_reg        | Flip-flop | 1024  |  Y  | N  | Y  | N  | N  | N  | N  |
|      valid2_reg       | Flip-flop |   8   |  Y  | N  | Y  | N  | N  | N  | N  |
|       tag1_reg        | Flip-flop |  200  |  Y  | N  | Y  | N  | N  | N  | N  |
|      valid1_reg       | Flip-flop |   8   |  Y  | N  | Y  | N  | N  | N  | N  |
|     chip_cen_reg      | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|     chip_wen_reg      | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|     chip_addr_reg     | Flip-flop |  30   |  Y  | N  | Y  | N  | N  | N  | N  |
=============================================================================================
Statistics for MUX OPs
```

Screenshot for performance

I0

```
============================================================

Success!
The test result is .....PASS :)
Total execution cycle : 78
============================================================

$finish called from file "../00_TB/tb.v", line 248.
$finish at simulation time                77500
```

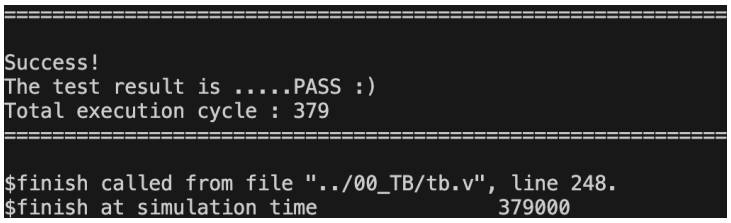
I0 (Cache)

```
============================================================

Success!
The test result is .....PASS :)
Total execution cycle : 59
============================================================

$finish called from file "../00_TB/tb.v", line 248.
$finish at simulation time                59000
```

I1

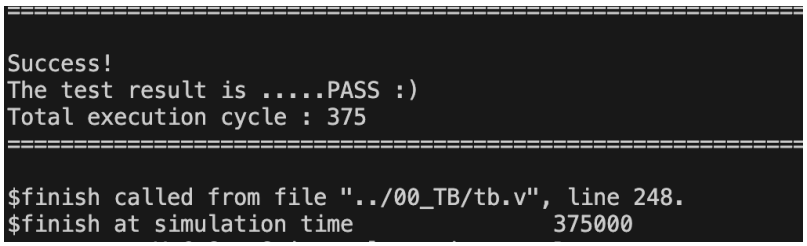
```
============================================================

Success!
The test result is .....PASS :)
Total execution cycle : 463
============================================================

$finish called from file "../00_TB/tb.v", line 248.
$finish at simulation time               462500
```

I1 (Cache)

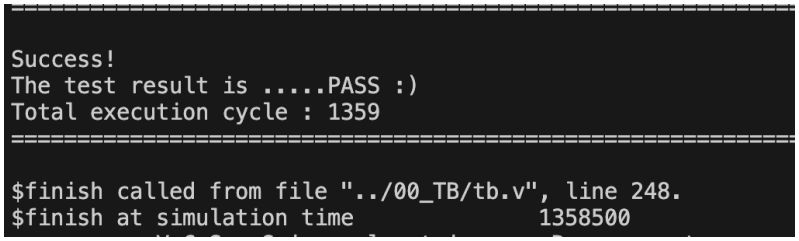


I2

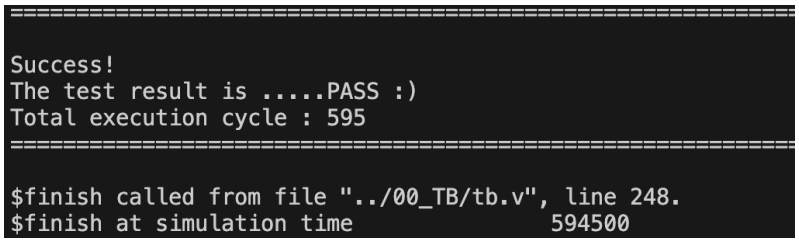
I2 (Cache)



I3



I3 (Cache)



Work distribution list

Cache: b10901158 and b10901091

Main chip: b10901158 and b10901091

report: b10901158 and b10901091