積體電路設計實驗 Integrated Circuit Design Laboratory

2023 FALL Final Project - Single Core Central Processing Unit (CPU)

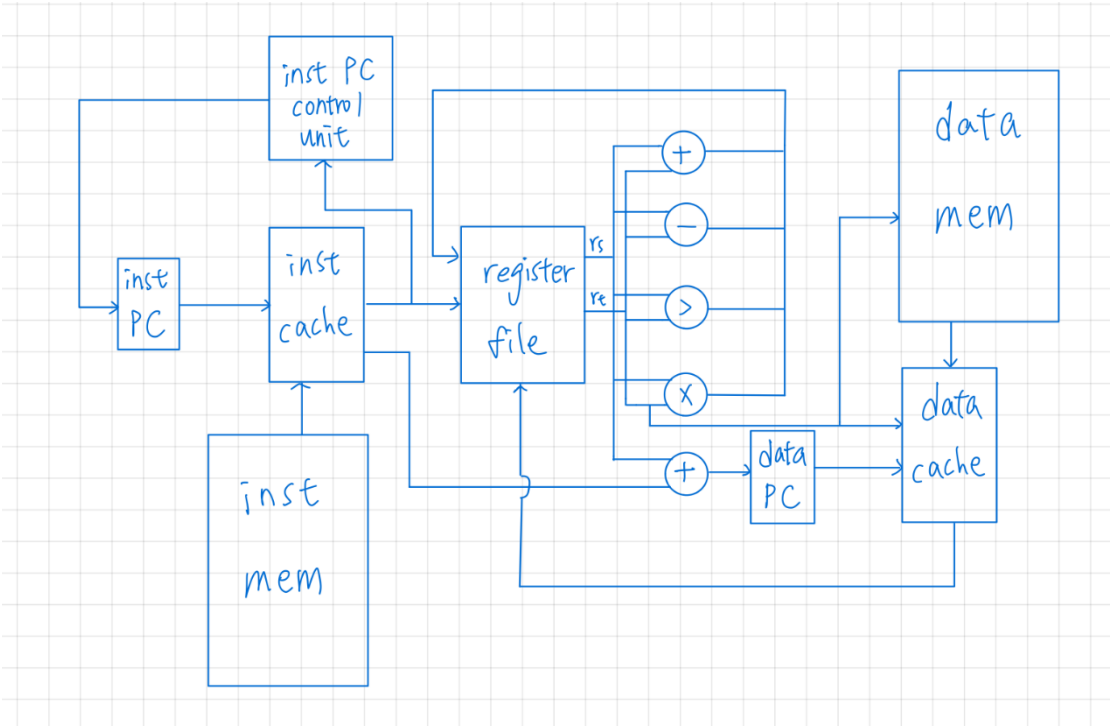
姓名:李品賢　帳號:iclab021　系級:電子碩 112　學號:312510151

一、　電路架構

硬體使用:

1. 128 words – 16 bits SRAM
   Instruction memory 的 cache
2. 128 words – 16 bits SRAM
   Data memory 的 cache
3. 16 bits 加法器
4. 16 bits 加法器 (signed)
5. 16 bits 減法器
6. 16 bits 比較器
7. 16 bits 乘法器

電路方塊圖:



圖一 電路方塊圖

二、 FSM (流程圖)

狀態使用:

1. IDLE

   電路的初始狀態，直接跳到 READ_INST_MEM state。

2. READ_INST_MEM

   當 instruction program counter 指到的 instruction 不在 instruction cache 中，需從 instruction memory 讀至 instruction cache，第 128 筆資料從 dram 送出後，跳到 WAIT_LAST_INST state。

3. WAIT_LAST_INST

   因為使用 DFF 將 dram 的資料送到 sram，因此必須多等一個 cycle 使最後一筆資料存入 sram，接著跳到 FETCH state。

4. FETCH

   將正確且經轉換後的 instruction program counter 送至 instruction cache，下一個 cycle，instruction cache (sram)會回傳對應的 instruction，接著跳到 DECODE state。

5. DECODE

   於此 state，instruction cache 送出對應的 instruction，且對 instruction 做解碼，存入 opcode, rs, rt, rd, func, immediate, jump_addr (registers)。同時，也將 rs, rt 對應的 core_reg 存入 rs_reg, rt_reg (registers)中，接著跳到 EXE state。

6. EXE

   於此 state，電路會根據 opcode，進行不同的運算。

   - R type:

     將 rs_reg, rt_reg 進行運算，存入 rd_reg；將 instruction program counter 加 2，接著跳到 WRITE_BACK state。

   - Load:

     將 rs_reg 和 immediate 進行運算，存入 data program counter；將 instruction program counter 加 2，接著跳到 BUF state。

   - Store:

     將 rs_reg 和 immediate 相加，存入 data program counter；將 instruction program counter 加 2，接著跳到 LOAD state。

   - Branch:

     若 rs_reg 和 rt_reg 相等，將 instruction program counter 和 immediate 運算後，存入 instruction program counter，若非，則將 instruction program

counter 加 2，接著跳到 CHECK_RANGE state。

- Jump:

  將 jump_addr 存入 instruction program counter，接著跳到 CHECK_RANGE state。

7. BUF

   於此 state，data program counter 才會更新成正確值，若電路第一次執行到 Load 指令，或 data cache 內沒有欲存取的資料，則跳到 READ_DARA_MEM state，反之，跳到 WAIT_LAST_DATA。

8. READ_DATA_MEM

   當 data program counter 指到的 data 不在 data cache 中，需從 data memory 讀至 data cache，第 128 筆資料從 dram 送出後，跳到 WAIT_LAST_DATA state。

9. WAIT_LAST_DATA

10. 此 state 有兩功能，第一，從 READ_DATA_MEM 來，則此 cycle 可以將最後存入 sram，第二，計算出欲讀取的 data cache 位置，接著跳到 WAIT_DARA_CACHE state。

11. CHECK_RANGE

    判斷由 Branch, Jump 產生的 instruction program counter 指到的位置，是否存在於 instruction cache 中，若成立，則跳到 FETCH state，若非，則跳到 READ_INST_MEM state

12. WAIT_DARA_CACHE

    將正確且經轉換後的 data program counter 送至 data cache，下一個 cycle，data cache (sram)會回傳對應的 data。

13. LOAD

    於此 state，data cache 送出對應的 data，且送入與 rs 對應的 core_reg。

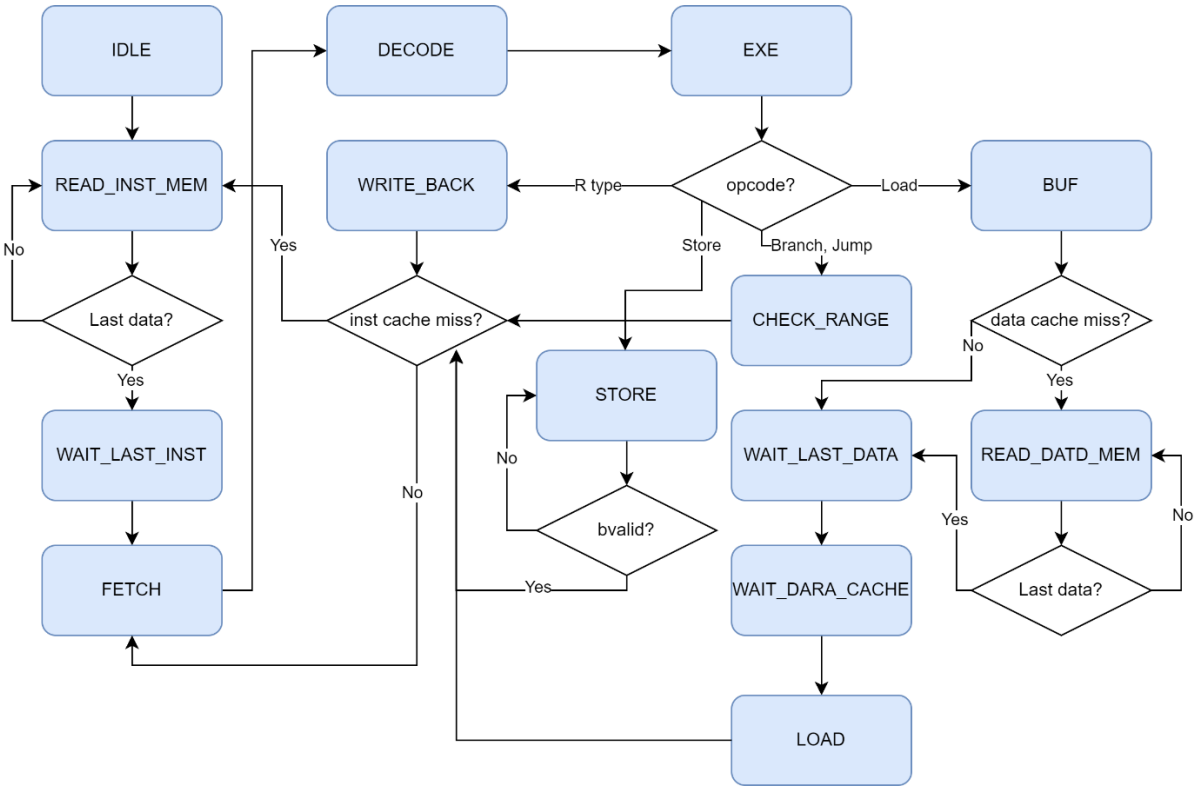
14. STORE

    於此 state，將 rt_reg 內的資料存入 data cache，也存入 data memory，若 bvalid_m_inf 為 1，則跳到 FETCH state。

15. WRITE_BACK

    將 rd_reg 存入與 rd 對應的 core_reg，且判斷 instruction program counter 指到的位置，是否存在於 instruction，若是，則跳到 FETCH state，若非，則跳到 READ_INST_MEM state。

Finite State Machine:



圖二 FSM 流程圖

三、 優化方法

1. 嘗試移除 critical path 的多於 mux，降低 cycle time

```
always @(posedge clk or negedge rst_n) begin
  if(!rst_n)
    rd_reg <= 0;
  else begin
    if(c_state == EXE) begin
      if(opcode == 3'b000) begin
        case(func)
          0: rd_reg <= rs_reg + rt_reg; // ADD
          1: rd_reg <= rs_reg - rt_reg; // SUB
        endcase
      end
      else if(opcode == 3'b001) begin
        case(func)
          0: rd_reg <= (rs_reg < rt_reg) ? 1 : 0; // SLT
          1: rd_reg <= rs_reg * rt_reg; // MULT
        endcase
      end
    end
  end
end
```

```
always @(posedge clk or negedge rst_n) begin
  if(!rst_n)
    rd_reg <= 0;
  else begin
    case({opcode[0], func})
      2'b00: rd_reg <= rs_reg + rt_reg; // ADD
      2'b01: rd_reg <= rs_reg - rt_reg; // SUB
      2'b10: rd_reg <= (rs_reg < rt_reg) ? 1 : 0; // SLT
      2'b11: rd_reg <= rs_reg * rt_reg; // MULT
    endcase
  end
end
```

右圖為左圖優化過的電路，經測試能使 cycle time 從 4.7 降至 3.9。

結果:

因左圖於 APR 階段能以 utilization 0.73 成功繞線，優化後的無法，因此還是選擇左圖方式。