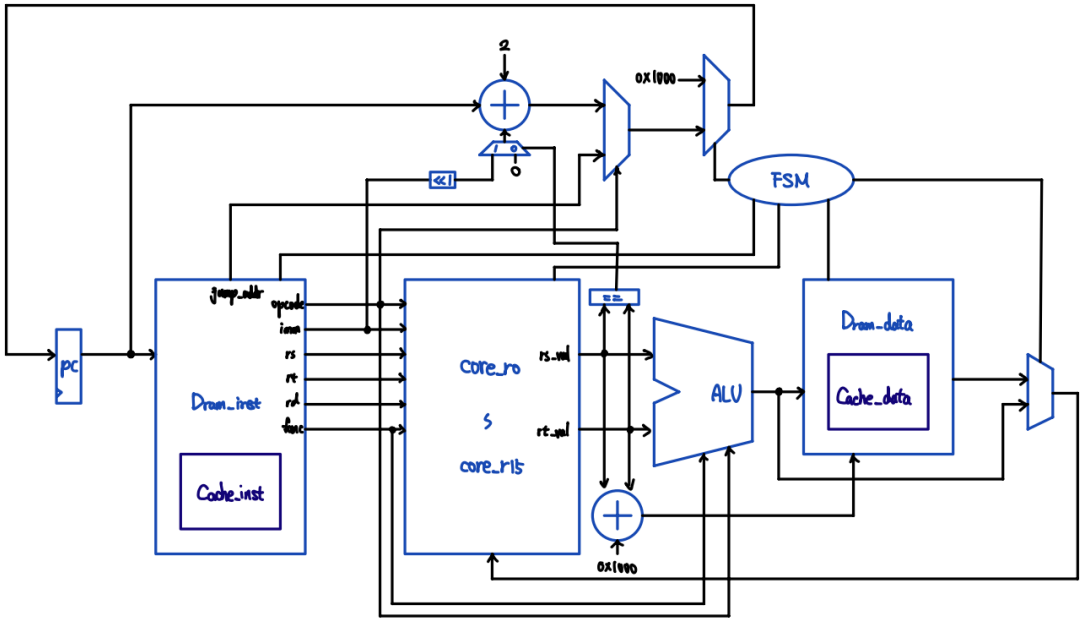
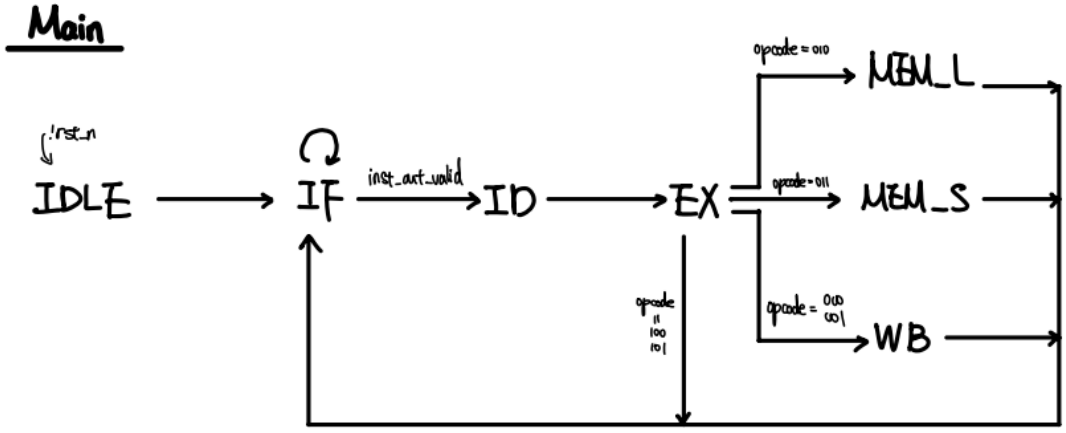
電路架構



上圖是這次 Final Project 一個簡略版的電路架構，裡面主要會有 PC、DRAM_inst、DRAM_data、ALU、Register File (core_r0～core_r15)、FSM 等單元。PC 主要管理目前 instruction 所在的位址，DRAM_inst 和 DRAM_data 則負責儲存 instruction 和 data。Register File 會用來存取目前 CPU 內部計算完的暫存值，ALU 負責算術運算，以及最後 FSM 負責管理整體電路的行為。

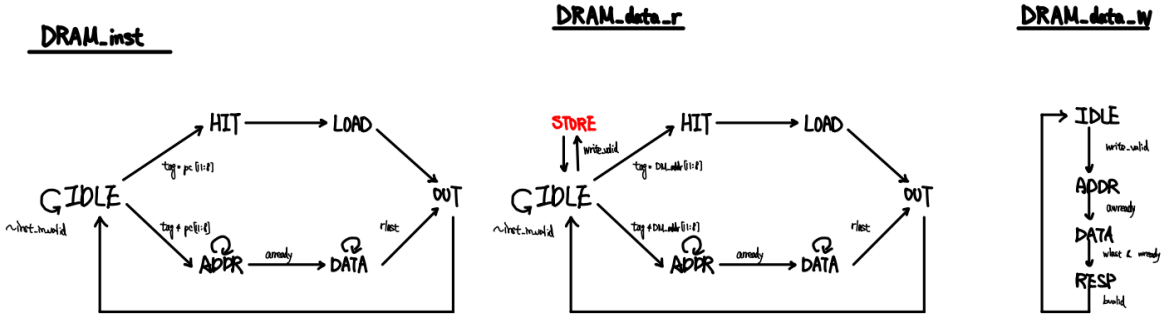
FSM

這次的狀態機主要分成四個部分：一個主要的狀態機，另外三個則是和 AXI 有關的狀態機。



如果把不同的 instruction 分開來看，可以得到：

```
ADD      : IF -> ID -> EXE -> WB
SUB      : IF -> ID -> EXE -> WB
SLT      : IF -> ID -> EXE -> WB
MULT     : IF -> ID -> EXE -> WB
LOAD     : IF -> ID -> EXE -> MEM_L
STORE    : IF -> ID -> EXE -> MEM_S
BOE      : IF -> ID -> EXE
JUMP     : IF -> ID -> EXE
```



在與 DRAM 之間使用的 AXI，可以分成上面三個狀態機，值得一提的是這裡加入了 CACHE，所以會多出 HIT 和 LOAD 的狀態來從 SRAM 裡面抓值出來，不用特地等 DRAM 來取值。

優化方法

1. 引入 CACHE 設計
   在這次的設計裡，DRAM_inst 與 DRAM_data 都是一次讀取 128 筆資料，並把他們都先放在 CACHE 裡面，如果下一次的讀取剛好在 CACHE 裡面，就可以直接從 CACHE 拿值，不用透過 DRAM 等待更長的時間來拿值。
   關於內部的細節，我使用的方法是利用 PC [11:8]當作 tag，PC[7:1]當作 CACHE 的 addr，如果當下要取的 DRAM addr 前四個位元與 PC[11:8]相同就表示 HIT，表示可以直接從 CACHE 拿值出來，反之就是 MISS，要花額外的 128 cycle 從 DRAM 拿值。

2. 使用 Multicycle
   與其把所有工作塞在一個 cycle 內完成，將其分成不同的 cycle 反而可能對整體的運作時間帶來更好的效益。因此這次的 Design 分別在 PC、DRAM_inst、ALU、DRAM_data 的輸出都裝了一個 DFF 以縮短 critical path，讓電路可以以更高的頻率去運作。

3. 針對 SRAM 輸入的 Hold time 問題
   在合成完 Gate netlist 的時候，發現在進行 03_GATE 的模擬時，SRAM 的輸入會發生 setuphold 的 timing violation，透過臉書社團上所獲得的資訊，在 SRAM 的輸入加上了一些 dummy mux 後就可以正確執行了。