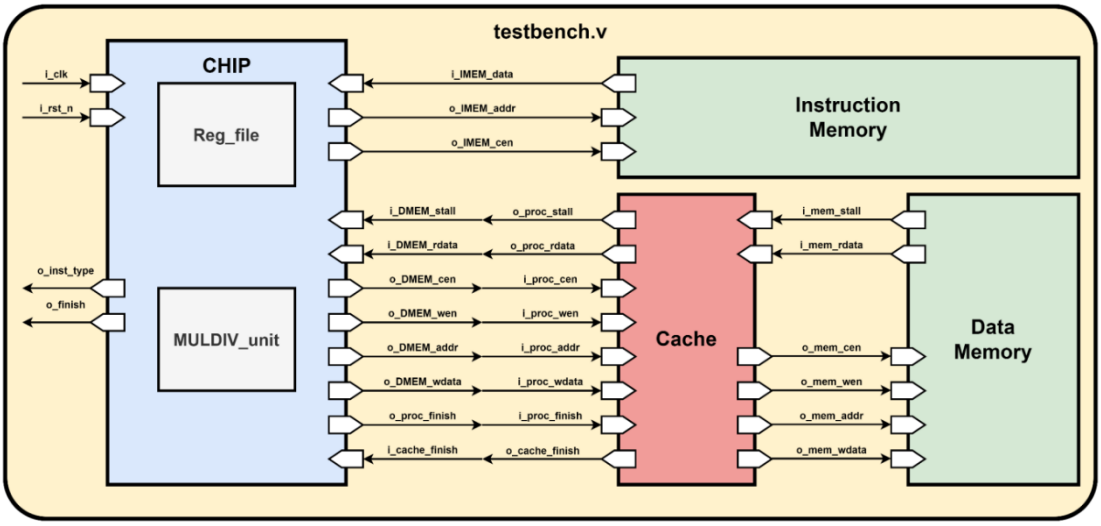
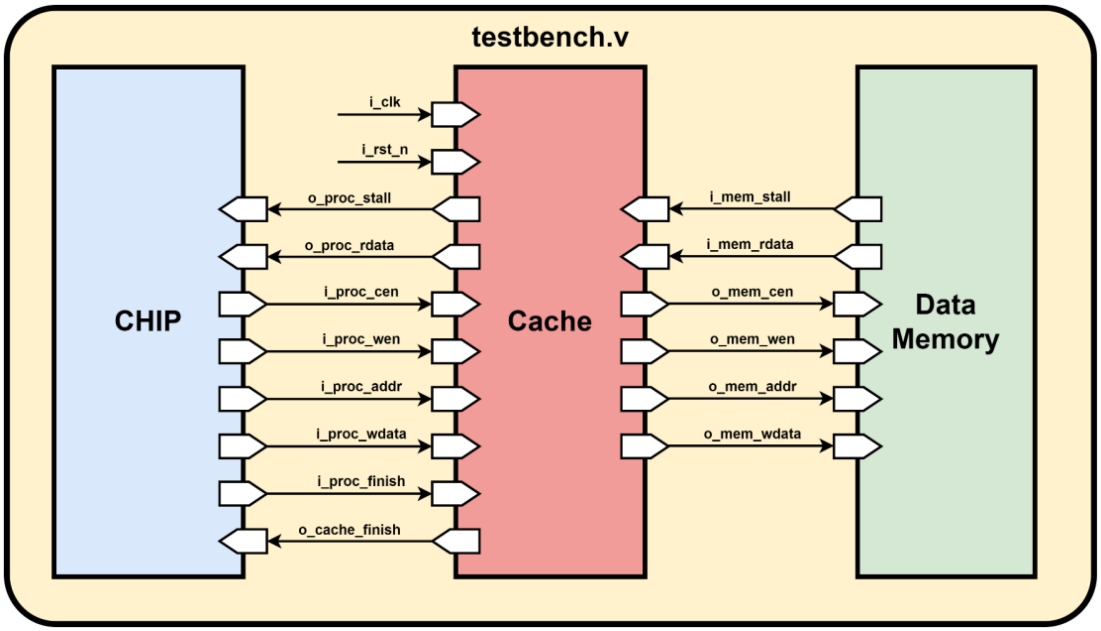
CA Final Project Report

Group 10
B09901080 吳宣逸 B09901084 陳威侑

Block Diagram (改編自投影片)



我們跟投影片的 block diagram 的差異是不使用 ALU，將除了 mul 之外的運算都直接放在 CHIP 進行，而不另外寫 submodule。



Instruction Implementation

- **auipc**: 從 instruction 裡取出 immediate 需要的 20 bits，後面補 0 形成 immediate。將 PC 的 next state 設為 PC 與 immediate 的相加(signed addition)，並將結果存入 rd

- **jal**: 從 instruction 裡取出 immediate 需要的 bits，經過 shift left 1 bit 和 sign extension 後形成 immediate。把目前 PC 加 4(下一個 instruction 的 address)存入 rd，將 PC 的 next state 設為 PC 與 immediate 的相加(signed addition)。

- **jalr**: 從 instruction 裡取出 immediate 需要的 bits

- **add**: 將 PC 的 next state 設為 PC 加 4，把 rs1 與 rs2 裡存的 value 經過 signed addition 後存入 rd。

- **sub**: 將 PC 的 next state 設為 PC 加 4，把 rs1 與 rs2 裡存的 value 經過 signed subtraction 後存入 rd。

- **and**: 將 PC 的 next state 設為 PC 加 4，把 rs1 與 rs2 裡存的 value 經過 bitwise and 後存入 rd。

- **xor**: 將 PC 的 next state 設為 PC 加 4，把 rs1 與 rs2 裡存的 value 經過 bitwise exclusive or 後存入 rd。-

- **addi**: 從 instruction 裡取出 immediate 需要的 bits，經過 sign extension 後形成 immediate，把 rs1 的 value 與 immediate 做 signed addition 後存入 rd，將 PC 的 next state 設為 PC 加 4。

- **slli**: 從 instruction 裡取出需要的 bits 得出 immediate，把 rs1 的 value，把 rs1 的 value shift left immediate，並存到 rd。將 PC 的 next state 設為 PC 加 4。

- **slti**: 從 instruction 裡取出需要的 bits 經過 sign extension 後得出 immediate。如果 rs1 的 value 小於 immediate 則將 1 存到 rd，如果不是則將 0 存到 rd。將 PC 的 next state 設為 PC 加 4。

- **srai**: 從 instruction 裡取出需要的 bits 得出 immediate，將 rs1 的 value 經過 arithmetic right shift immediate 後存到 rd。將 PC 的 next state 設為 PC 加 4。

- **lw**: 從 instruction 裡取出需要的 bits 經過 sign extension 得出 immediate。將 rs1 裡的 value 和 immediate 做 signed addition 後得出要 load 的 address。若是 data memory stall，則 PC 維持不變，若是 data memory 沒有在 stall(已經拿到 value)，則將 PC 的 next state 設為 PC 加 4。

- **sw**: 從 instruction 裡取出需要的 bits 經過 sign extension 得出 immediate。將 rs1 裡的 value 和 immediate 做 signed addition 後得出要 store 的 address。把 rs2 裡的 value 存到這個 address。若是 data memory stall，則 PC 維持不變，若是 data memory 沒有在 stall(已經存入 value)，則將 PC 的 next state 設為 PC 加 4。

- **mul**: 利用 HW2 的架構做出一個 submodule(MULDIV_unit)，把訊號(muldiv_valid)設成 1 一個 cycle 並傳進 MULDIV_unit 通知 submodule 開始進行運算，並等待 submodule 完成運算。Submodule 完成運算後會傳出訊號(muldiv_done)告知完成。若是 muldiv_done 則把 PC 的 next state 設為 PC 加 4、算出的結果存進 rd。若沒有則維持 PC 不變。

- **beq**: 從 instruction 裡取出需要的 bits 經過 sign extension 得出 immediate，若是 rs1 的 value 與 rs2 的 value 相同，則跳轉到 PC+immediate shift left 1 bit(signed addition)。若不是則跳轉到 PC+4。

- **bge**: 從 instruction 裡取出需要的 bits 經過 sign extension 得出 immediate，若是 rs1 的 value 大於等於 rs2 的 value，則跳轉到 PC+immediate shift left 1 bit(signed addition)。若不是則跳轉到 PC+4。

- **blt**: 從 instruction 裡取出需要的 bits 經過 sign extension 得出 immediate，若是 rs1 的 value 小於 rs2 的 value，則跳轉到 PC+immediate shift left 1 bit(signed addition)。若不是則跳轉到 PC+4。

- **bne**: 從 instruction 裡取出需要的 bits 經過 sign extension 得出 immediate，若是 rs1 的 value 不等於 rs2 的 value，則跳轉到 PC+immediate shift left 1 bit(signed addition)。若不是則跳轉到 PC+4。

- **ecall**: 將 o_finish 設為 1，通知 cache 進入 S_CLEAN。

Handling Multi-Cycle Instructions

利用 HW2 的架構做出一個 submodule(MULDIV_unit)，此 submodule 負責 multi-cycle 的 operation，當需要它進行運算時把訊號(muldiv_valid)設成 1 一個 cycle 並傳進 MULDIV_unit 通知 submodule 開始運算，並等待 submodule 完成運算。submodule 完成運算後會傳出訊號(muldiv_done)告知上層完成。若是 muldiv_done 則把 PC 的 next state 設為 PC 加 4、算出的結果存進 rd。若沒有則維持 PC 不變。因此在 MULDIV_unit 尚未完成運算時 CHIP 會等它。

Observation

因為沒有做 pipeline CPU，也無法調整一個 cycle 的時間，所以提升 time performance 的關鍵是減少 multi-cycle instruction(sw、lw、mul)所需要的 cycle。mul 因為無法使 MULDIV_unit 變得更快，所以加速的重點擺在 sw、lw 上，利用 cache 的 locality 提升 time performance。

Cache Architecture

我們實做的是 write-back direct-mapped cache with write allocation。每個 block 大小是 128 bits 包含 4 個 32-bit entry，整個 cache 共有 16 個 blocks，對應的 address 如下：

31:8	7:4	3:2	1:0
tag	index	block offset	byte offset

關於 address 的處理，由於 DMEM 有最小 address 的限制：i_offset，而且 cache 一次會取 4 個 word 放進 cache 的 block (128 bits)，為了方便 parse address，會希望這 4 個 word 的 address 是

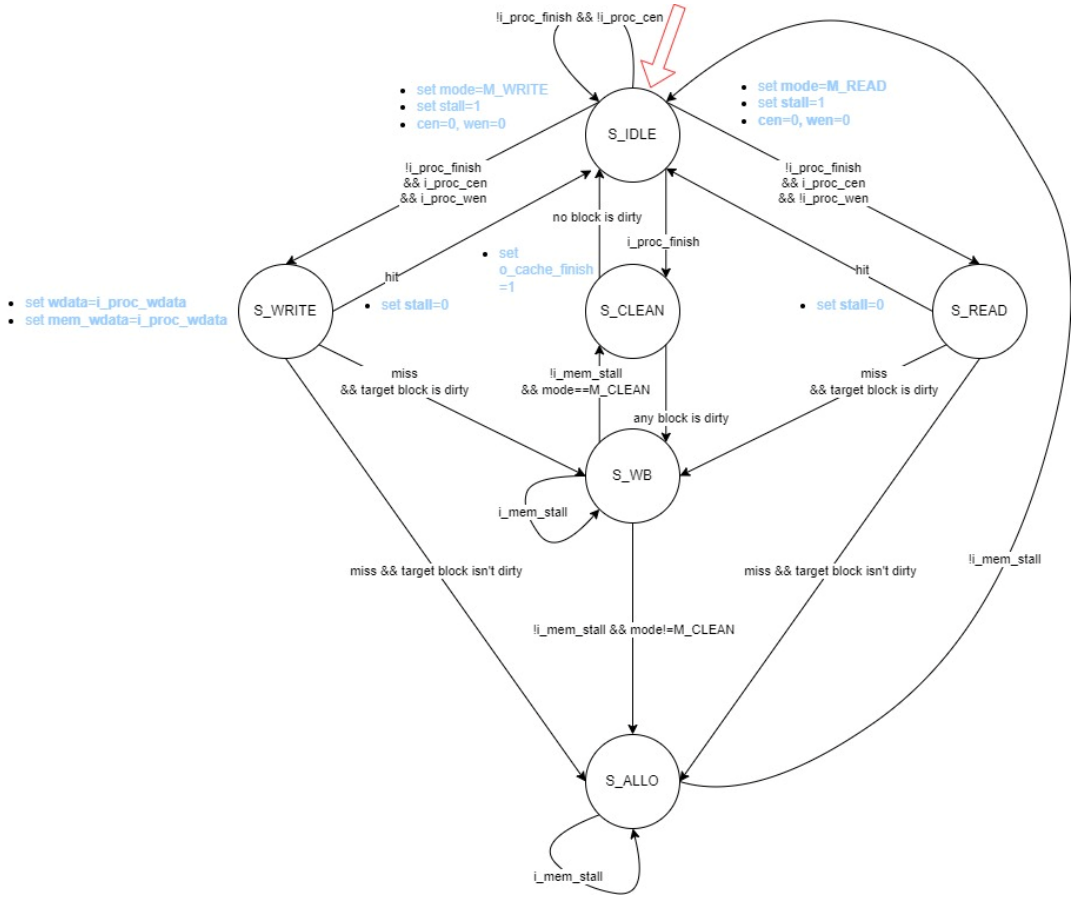
0x????_???**0**	0x????_???**4**	0x????_???**8**	0X????_???**c**

，並且包含要使用的 word，如此一來可能造成 cache 想要取到的 4 個 word 中有 DMEM 中不存在的 address。例如 i_offset=0x2013_1224，若是 address 是 0x2013_1228 且使用原本的 address 對應 index，則該 block 會包含不存在的 address：0x2013_1220，造成結果錯誤或浪費 cache 的空間。因此我們額外定義 cache 使用的 address 為 real_address：

$$real\ address = address - i\_offset$$

由於 real_address 與 DMEM 中實際標記位置的 index 互相對應，意味著 cache 使用的 address (real_address)從 0x0000_0000 開始編號，完美避免以上提到的問題。

我們使用一個具有 IDLE state 的 FSM 實作這個 cache，與講義中的 FSM 非常類似 (Ch5 p.96)，有 write back (S_WB)與 allocation (S_ALLO)兩種處理 miss 的狀況，S_READ、S_WRITE 則是對應於講義中的”compare tag”。不同的是，我們增加 S_CLEAN 的 state，在 procedure 結束時將 cache 與 DMEM 同步，具體 FSM 如下：



簡單來說，此 FSM 須要處理的情形大概有以下三大類：

- Write (S_WRITE)：將對應 block 的 dirty bit 設為 1
  - Hit：直接寫入 cache
  - Miss：若 block 是 dirty 則先寫回 DMEM (S_WB)，否則直接從 DMEM 將 data 載入到 cache 並同時寫入目標 entry (S_ALLO)
- Read (S_READ)
  - Hit：直接讀取 cache
  - Miss：若 block 是 dirty 則先寫回 DMEM 並將 dirty bit 設為 0 (S_WB)，否則直接從 DMEM 將 data 載入到 cache 並同時讀取 (S_ALLO)
- Clean (S_CLEAN)：

  cache 收到 i_proc_finis 的信號時 (對應 ecall)，會進入此 state 逐一將所有 dirty block 寫回 DMEM (S_WB)並將 dirty bit 還原成 0，直到所有 dirty bit=0 (clean)。

  藉由 write back with write allocation 的機制，read 和 write 使用 DMEM 的時間都有

機會減短，儘管 write 多了將 memory 的 data 載入 cache 的步驟而多了一次 memory accessing，但因為一個 block 有 4 個 word 並且其中不乏頻繁操作 stack。在呼叫的 address 非常連續的情形下，儘管考慮以上狀況以及額外的 state transition 占用的 clock cycle，write back with write allocation 的機制仍足以有效減少 clock cycle。由以下的數據可看出在 testbench I3 甚至有 2.49 的 speedup。

Time performance improvement with cache

Instruction set	Without Cache	With Cache	Speedup
I0	78	67	1.16
I1	463	382	1.21
I2	459	438	1.05
I3	1359	545	2.49

Register Tables

1. Cache: 總共 2761 個 registers

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
entry_reg	Flip-flop	2048	Y	N	Y	N	N	N	N
dirty_reg	Flip-flop	16	Y	N	Y	N	N	N	N
v_reg	Flip-flop	16	Y	N	Y	N	N	N	N
state_reg	Flip-flop	3	Y	N	Y	N	N	N	N
mode_reg	Flip-flop	1	N	N	Y	N	N	N	N
mode_reg	Flip-flop	1	N	N	N	Y	N	N	N
addr_reg	Flip-flop	32	Y	N	N	N	N	N	N
rdata_reg	Flip-flop	32	Y	N	Y	N	N	N	N
wdata_reg	Flip-flop	32	Y	N	Y	N	N	N	N
mem_wdata_reg	Flip-flop	128	Y	N	Y	N	N	N	N
mem_addr_reg	Flip-flop	32	Y	N	N	N	N	N	N
real_addr_reg	Flip-flop	32	Y	N	Y	N	N	N	N
stall_reg	Flip-flop	1	N	N	Y	N	N	N	N
finish_reg	Flip-flop	1	N	N	Y	N	N	N	N
cen_reg	Flip-flop	1	N	N	Y	N	N	N	N
wen_reg	Flip-flop	1	N	N	Y	N	N	N	N
tag_reg	Flip-flop	384	Y	N	Y	N	N	N	N

2. CHIP

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
DMEM_cen_reg	Flip-flop	1	N	N	Y	N	N	N	N
DMEM_wen_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
rdatad_reg	Flip-flop	32	Y	N	Y	N	N	N	N
DMEM_addr_reg	Flip-flop	32	Y	N	Y	N	N	N	N
DMEM_wdata_reg	Flip-flop	32	Y	N	Y	N	N	N	N
finish_reg	Flip-flop	1	N	N	Y	N	N	N	N
muldiv_ready_reg	Flip-flop	1	N	N	Y	N	N	N	N
DMEM_ready_reg	Flip-flop	1	N	N	N	Y	N	N	N

3. Reg_file

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

4. MULDIV_unit

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
cnt_nxt_reg	Flip-flop	5	Y	N	N	N	N	N	N

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
o_data_cur_reg	Flip-flop	64	Y	N	Y	N	N	N	N
o_done_cur_reg	Flip-flop	1	N	N	Y	N	N	N	N
state_reg	Flip-flop	2	Y	N	Y	N	N	N	N
operand_a_reg	Flip-flop	32	Y	N	Y	N	N	N	N
operand_b_reg	Flip-flop	32	Y	N	Y	N	N	N	N
cnt_reg	Flip-flop	5	Y	N	Y	N	N	N	N

Work Distribution

- B09901080 吳宣逸：
  1. Cache implementation
  2. MULDIV_unit bug fix
  3. Report (cache architecture)
- B09901084 陳威侑：
  1. CHIP implementation
  2. MULDIV_unit implementation
  3. Report (block diagram、instructions implementation、handling multi-cycle instructions、observation)