HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL & ELECTRONICS ENGINEERING

REPORT DESIGN PROJECT

RVS192

30.5.2020

Lê Quang Hưng

1711631

MỤC LỤC

LỜI NÓI ĐẦU

Đây là tài liệu mô tả hoạt động, chức năng hỗ trợ và cách cấu hình tài nguyên sử dụng và cấu trúc của lõi RVS192

Phát triển bởi Lê Quang Hưng – ĐH Bách Khoa HCM

PHẦN MỀM SỬ DỤNG

Phần mềm	Mục đích sử dụng
Microsoft Word 2016	Viết báo cáo, mô tả cấu trúc
drawio	Vẽ sơ đồ cấu trúc
Notepad++	Mô tả RTL code
QuestaSim	Mô phỏng RTL code
Quartus Prime Lite Edition 18.1	Khả tổng hợp RTL code
RARS 1.0	Tạo assembly code cho RVS192

THUẬT NGỮ VÀ VIẾT TẮT

RVS192: RISC V semester 192

RISC: Reduced Instruction Set Computer

BPU: Branch Prediction Unit

BHT: Branch History Table

BTB: Branch Target Buffer

QUY ĐỊNH ĐẶT TÊN CÁC TÍN HIỆU

Các tín hiệu ngõ ra có tên giống nhau (cùng đặc điểm) trong cùng một tầng Pipeline sẽ được thêm tiền tố là các chữ cái hoa, viết tắt tên của khối xuất dữ liệu tương ứng.

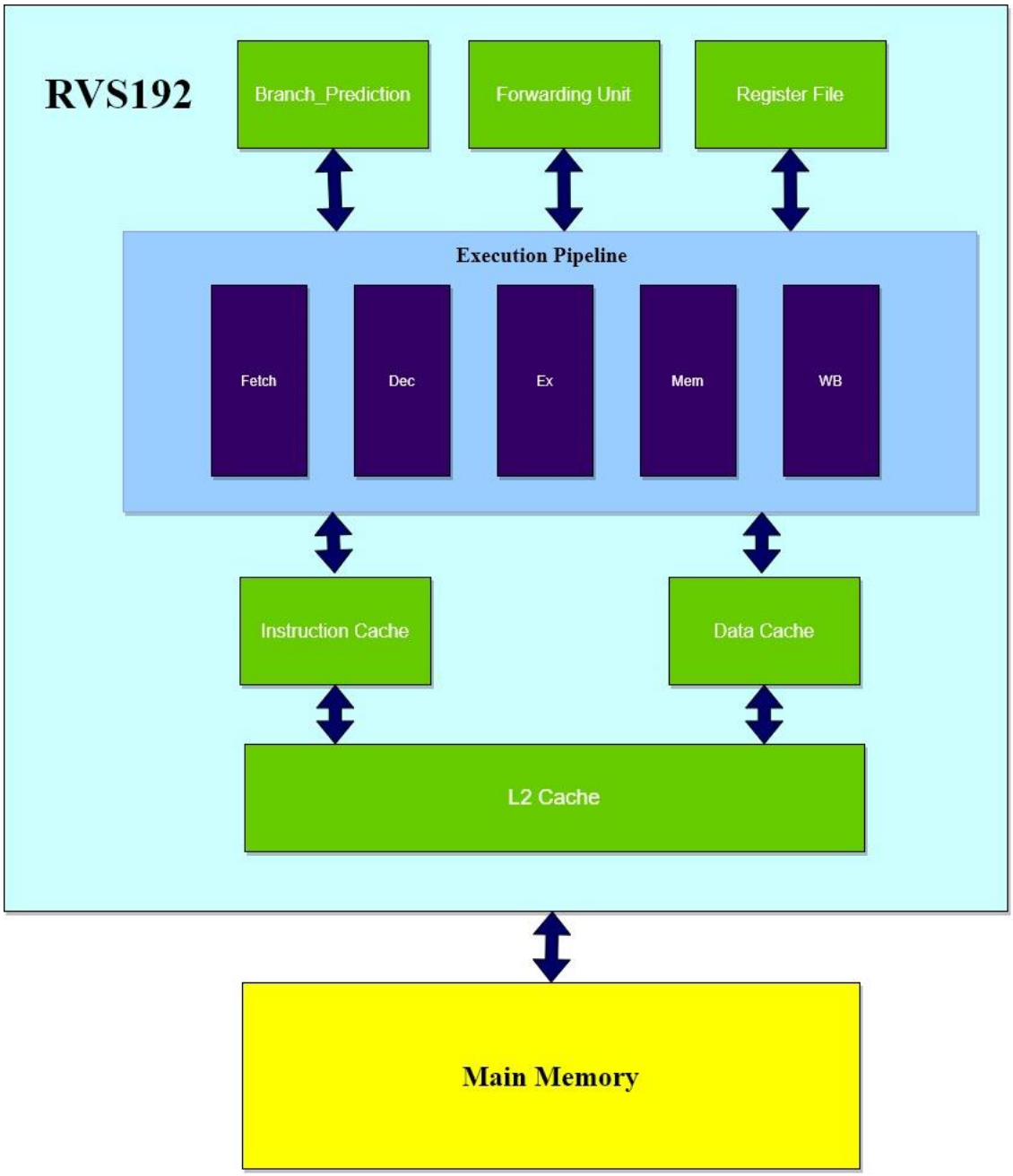
Ví dụ: IL1_hit, IVC_hit,…

Các tín hiệu được truyền đi qua các tầng Pipeline sẽ được thêm hậu tố là tên của tầng đó

Ví dụ: pc_fetch, pc_decode, …

1 ĐẶC ĐIỂM THIẾT KẾ

1.1 GIỚI THIỆU



Hình 1.1: RVS192 architecture

RVS192 là một lõi cpu được thiết kế theo kiến trúc tập lệnh RV32I, với cấu trúc đơn nhân, đơn lệnh (single-core, single-issue). Lõi RVS192 được phát triển bằng ngôn ngữ mô tả phần cứng SystemVerilog.

RVS192 sử dụng cấu trúc Harvard để hiện thực việc truy cập Instruction và Data từ bộ nhớ. Lõi cpu trên được thiết kế thành 5 tầng Pipeline nhằm giảm công suất tiêu thụ và tăng tần số hoạt động. RVS192 tích hợp thêm bộ Branch Prediction nhằm giảm số chu kỳ dừng của cpu , giúp cải thiện hiệu suất (CPI). Ngoài ra, các bộ Instruction Cache, Data Cache và L2 Cache được thêm vào để giảm thời gian truy cập dữ liệu từ bộ nhớ, nhằm ép tần số hoạt động của RVS192 cpu lên cao hơn nữa.

Lõi RVS192 hỗ trợ người dùng thay đổi các thông số như organization, size, associativity của các bộ Cache và cấu hình lại bộ Branch Prediction. Hỗ trợ này được thêm vào nhằm giúp người dùng tùy chỉnh (trade-off) giữa các thông số như hiệu suất, năng lượng và tài nguyên sử dụng cho lõi RVS192 trong các ứng dụng khác nhau.

Người dùng có thể xem thêm về kiến trúc tập lệnh RV32I được lõi RVS192 hỗ trợ trong tài liệu ***The RISC-V Instruction Set Manual Version 2.2 được công bố bởi RISC-V Foundation (https://riscv.org/).***

1.2 ĐẶC ĐIỂM HỖ TRỢ

RVS192:

- 32-bit instruction set RV32I
- 5-stage pipeline + forwarding unit
- Optional/Parameterized Branch-Prediction Unit
- Optional/Parameterized Caches

Các thông số và phần cứng có thể cấu hình

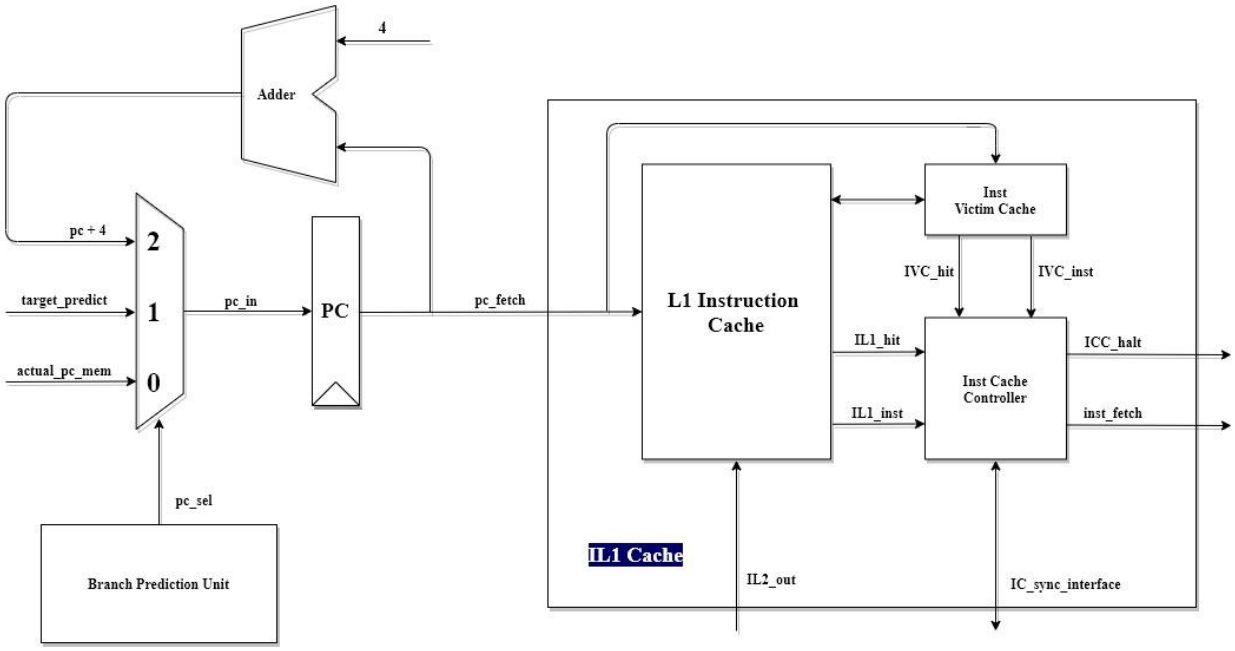
- Cache Size/organization/associativity
- User selectable Branch Prediction Unit

Tối ưu công suất tiêu thụ

- Gated clock design

2 CÁC TẦNG PIPELINE

2.1 FETCH STAGE



Hình 2.1: Fetch Stage

Tầng Fetch làm nhiệm vụ truy suất bộ nhớ lệnh để lấy câu lệnh cần thực hiện.

Tín hiệu pc_sel được gửi tới từ bộ BPU, giúp chọn địa chỉ nhảy đến của lệnh kế tiếp.

- pc_sel = 2'b00: pc_in = actual_pc, đây là địa chỉ thực sự nhảy đến sau lệnh nhảy, đường này sẽ được chọn khi BPU phát hiện ra rằng nó đã đoán sai.
- pc_sel = 2'b01: pc_in = target_predict, đây là địa chỉ dự đoán từ bộ BPU, đường này sẽ được chọn khi BPU gặp một lệnh nhảy.
- pc_sel = 2'b01: pc_in = pc+4, đường này sẽ được chọn khi BPU không gặp lệnh nhảy

Địa chỉ pc_fetch được gửi đến khối L1 Instruction Cache và Inst Victim Cache để kiểm tra xem IL1 Cache có chứa câu lệnh cần fetch hay không. Nếu có thì fetch câu lệnh (inst_fetch) và kéo tín hiệu ICC_halt xuống mức thấp. Nếu không, Inst Cache Controller

sẽ tích cực tín hiệu ICC_halt để yêu cầu dừng Pipeline chờ câu lệnh được fetch từ các tầng memory chậm hơn (L2 cache, main memory). Xem mục 7. CACHE

Bộ BPU sẽ được mô tả rõ hơn ở mục 6. BRANCH PREDICTION UNIT

Danh sách các tín hiệu cần được truyền sang tầng tiếp theo:

Tên tín hiệu	Mô tả
pc_fetch	pc được lấy ra từ tầng Fetch
br_check_fetch	Tổ hợp các tín hiệu kiểm tra tính đúng đắn của lệnh nhảy ở pha Fetch
Inst_fetch	Instruction được lấy ra từ tầng Fetch

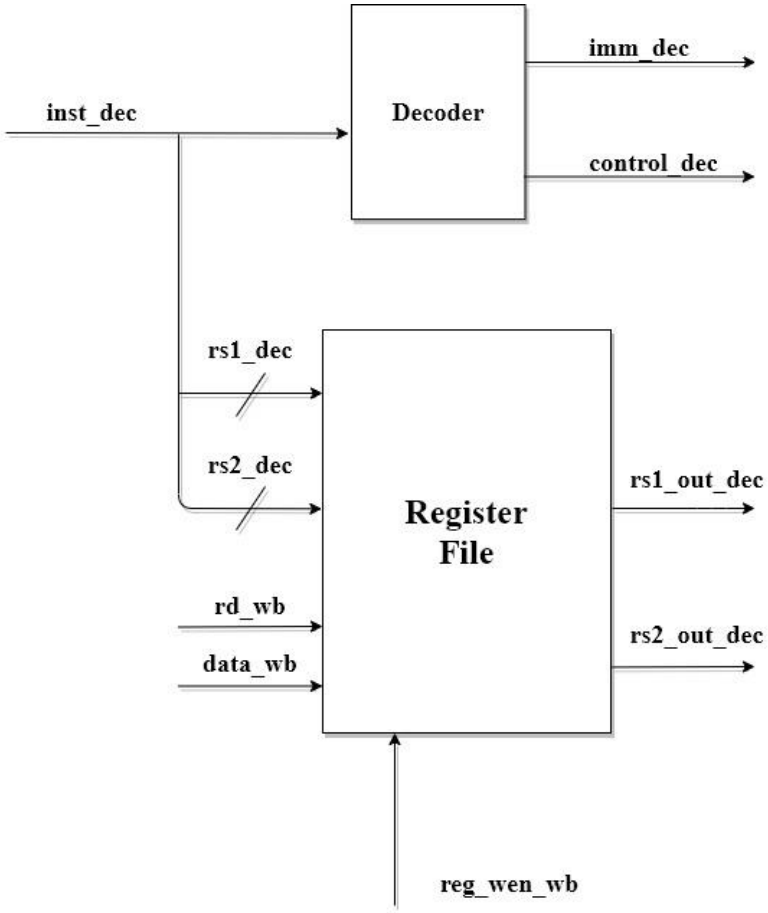
Bảng 2.1: Bảng mô tả tín hiệu từ tầng Fetch, cần gửi đến các tầng sau

2.2 DECODE STAGE

Tầng Decode thực hiện nhiệm vụ giải mã lệnh để đưa ra các tín hiệu điều khiển và dữ liệu cho các tầng tiếp theo xử lý.

Khối Register File chứa 32 thanh ghi 32 bit (x0-x31). Trong đó có các thanh ghi dùng chung, nhằm lưu các toán hạng hay giá trị sau khi tính toán. Ngoài ra còn có một số thanh ghi đặc biệt như zero(x0), ra(x1), sp(x2), gp(x3), tp(x4).

Khối Controller sẽ giả mã các câu lệnh để đưa ra các tín hiệu điều khiển gửi tới các tầng sau. Bên cạnh đó, Controller còn dùng để tạo giá trị immediate theo từng loại câu lệnh tương ứng.



Hình 2.2: Decode Stage

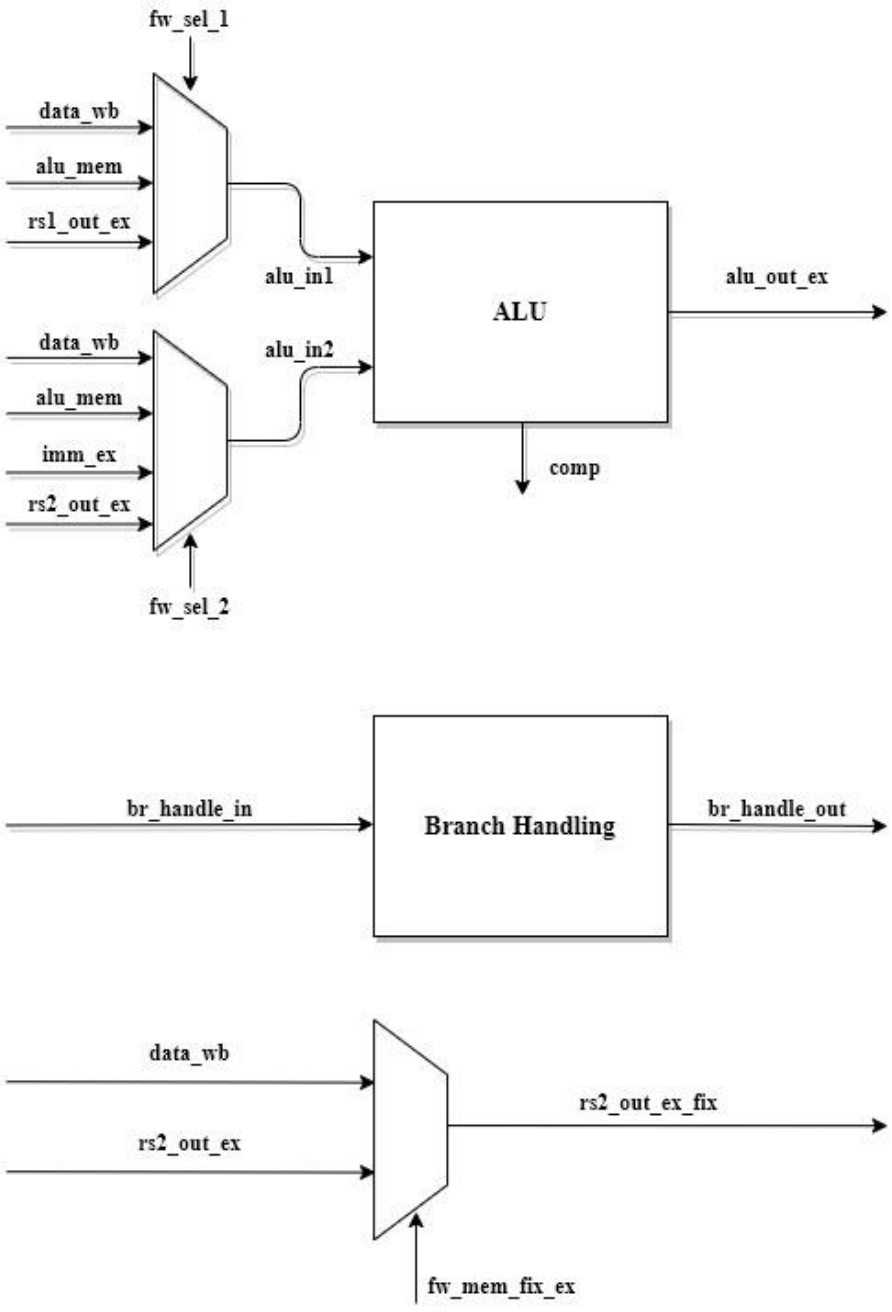
Danh sách các tín hiệu cần được truyền sang tầng tiếp theo:

Tên tín hiệu	Mô tả
pc_dec	pc ở tầng Decode
br_check_dec	Tổ hợp các tín hiệu kiểm tra tính đúng đắn của lệnh nhảy ở pha Decode
reg_addr_dec	Tổ hợp địa chỉ của các thanh ghi nguồn và đích ở tầng Decode
reg_out_dec	Tổ hợp giá trị của 2 thanh ghi nguồn ở tầng Decode

decoder_out_dec	Tổ hợp tín hiệu immediate và control được tạo ra từ bộ Decoder

Bảng 2.1: Bảng mô tả tín hiệu từ tầng Decode, cần gửi đến các tầng sau

2.3 EXECUTE STAGE

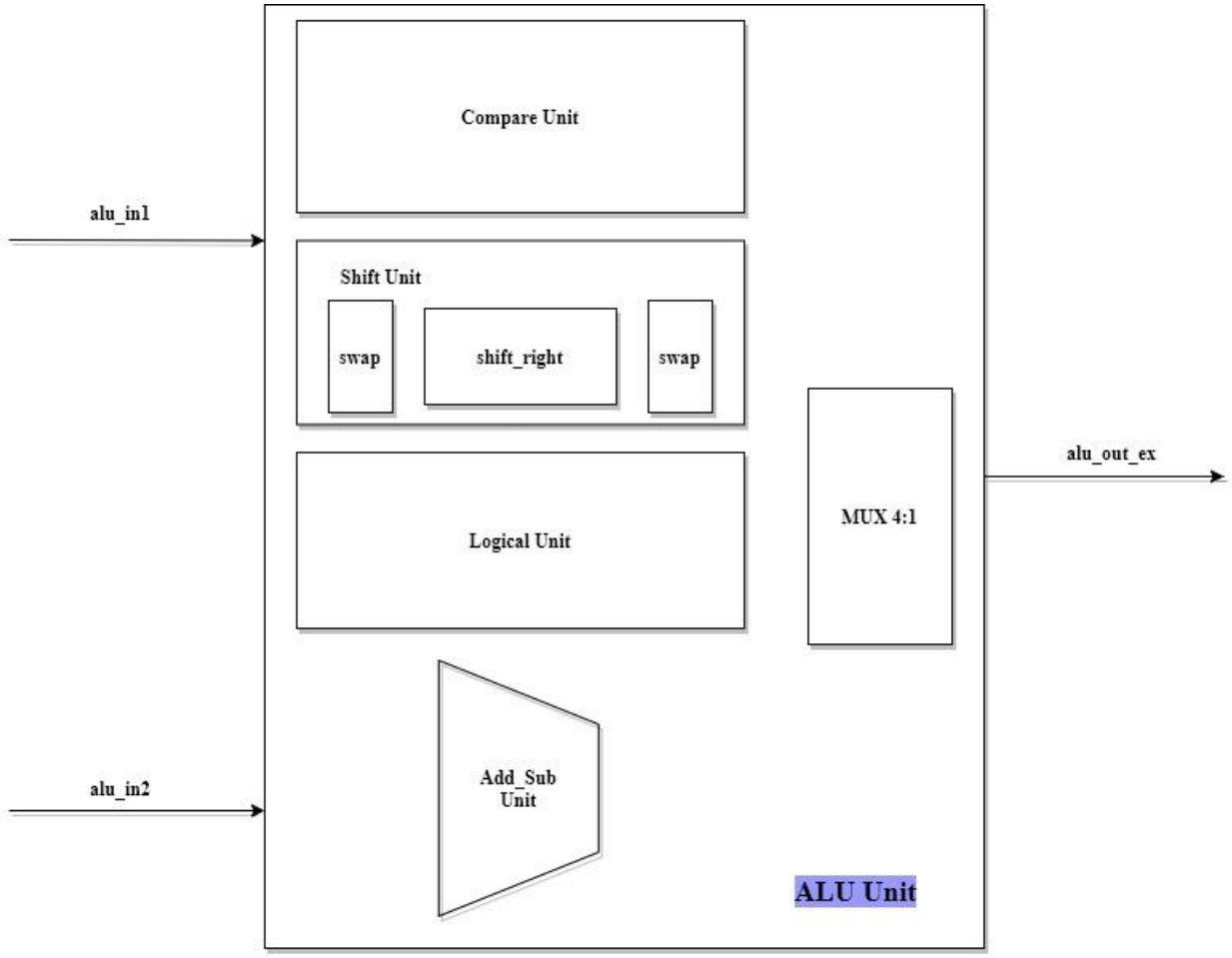


Hình 2.3.1: Execute Stage

Tầng Execute thực thi các tác vụ so sánh, dịch bit, tính toán logic và số học, tính toán địa chỉ và kiểm tra tính đúng đắng của lệnh nhảy.

Bộ ALU được tối ưu để tầng Execute sử dụng ít tài nguyên nhất có thể:

- Compare Unit: so sánh lớn hơn hoặc bằng
- Add_Sub Unit cộng trừ nhị phân
- Shift right: dịch phải
- Logical Unit: thực hiện các phép toán logic: |, &, ^



Hình 2.3.2: ALU Unit

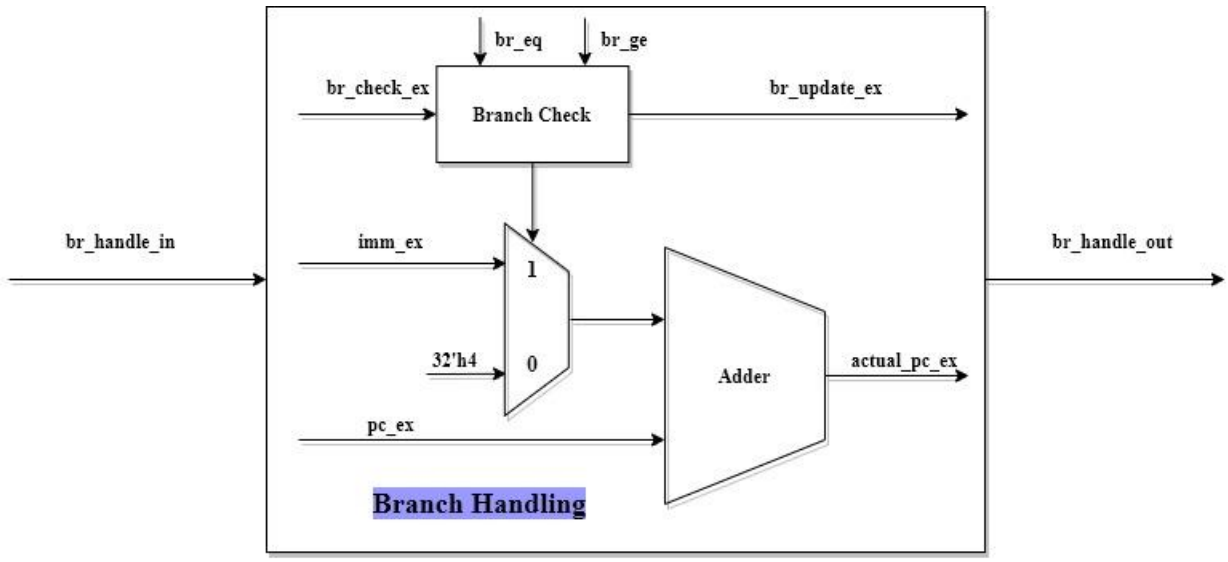
Dưới đây là bảng mô tả các câu lệnh sử dụng tài nguyên của bộ ALU

Tài nguyên của ALU	Các câu lệnh sử dụng tài nguyên của ALU	alu_op	
		2 bit cao	2 bit thấp

Add_Sub Unit	add/addi lb/lh/lw/lbu/lhu jal/jalr lui	00	00
	Sub	00	01
Shift Unit	sll/slli	01	00
	srl/srli	01	01
	sra/srai	01	10
Logical Unit	Or	10	00
	And	10	01
	Xor	10	10
Compare Unit	slt/slti beq bne blt bge	11	00
	sltu/sltiu bltu bgeu	11	01

Bảng 2.3.1: Bảng mô tả các lệnh sử dụng tài nguyên ALU

Bộ Branch Handling chứa một bộ cộng để tính địa chỉ nhảy đến của lệnh nhảy. Ngoài ra, nó còn chứa bộ Branch Check giúp kiểm tra tính đúng đắng của lệnh nhảy và gửi các thông tin update sửa lỗi cho bộ BPU.



Hình 2.3.3: Branch Handling

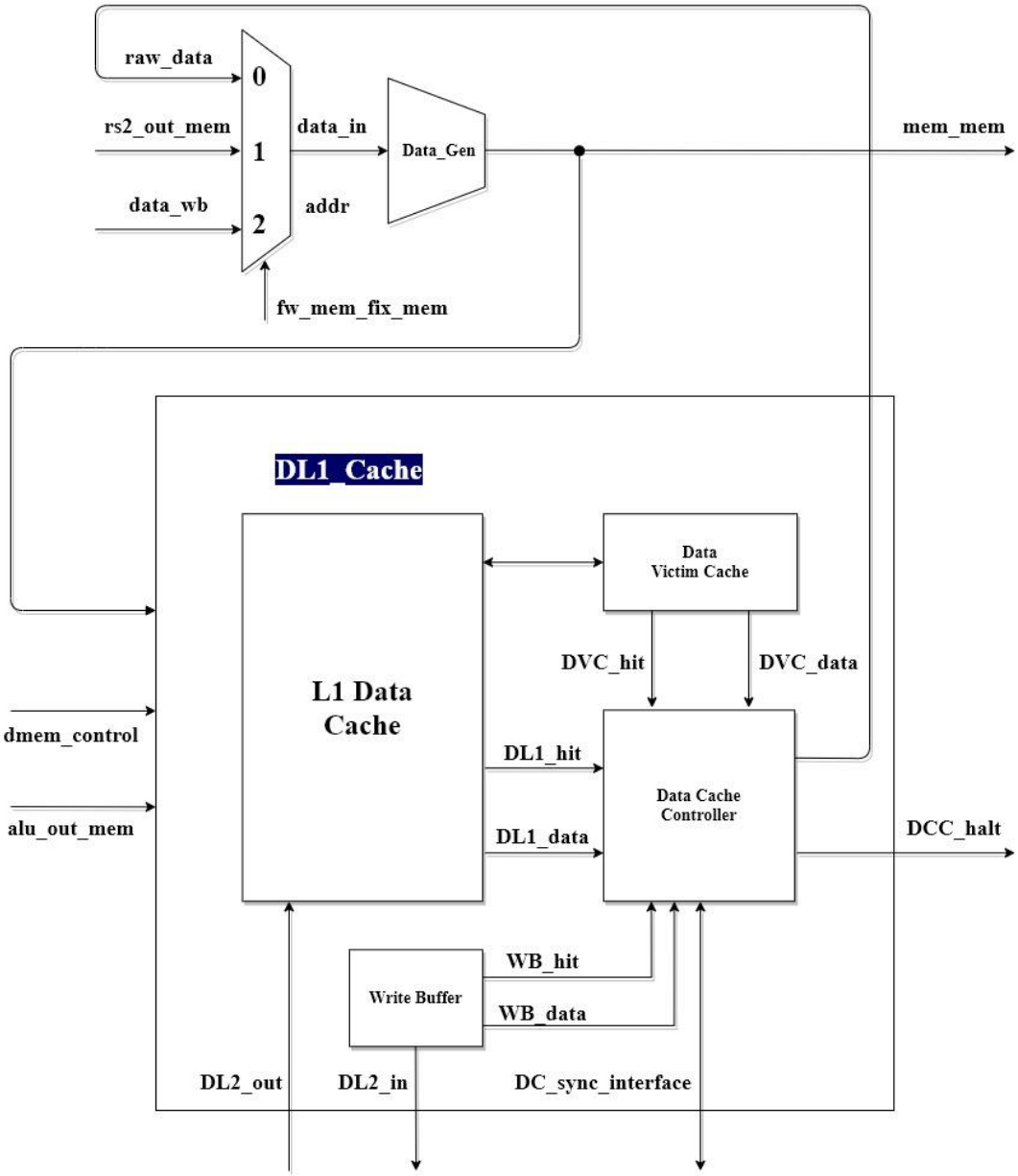
Các bộ mux trên sơ đồ cấu trúc của tầng Execute được điều khiển bởi bộ Forwarding Unit, nhằm sửa các lỗi data hazard ở tầng Execute . Xem mục 3. FORWARDING UNIT

Danh sách các tín hiệu cần được truyền sang tầng tiếp theo:

Tên tín hiệu	Mô tả
rd_addr_ex	Địa chỉ thanh ghi đích ở tầng Execute
br_handle_out.actual_pc_ex	Lệnh nhảy: Địa chỉ nhảy đến thực sự của lệnh nhảy Lệnh khác: PC+4
br_handle_out .wrong_ex	Tín hiệu sửa lỗi cho lệnh nhảy
alu_out_ex	Kết quả từ bộ ALU
rs2_out_ex_fix	Dữ liệu được forwarding trước khi ghi vào DMEM
decoder_out_ex	Tổ hợp tín hiệu control cần gửi đến tầng Mem

Bảng 2.3.2: Bảng mô tả tín hiệu từ tầng Execute, cần gửi đến các tầng sau

2.4 MEM STAGE



Hình 2.4: Mem Stage

Tầng Mem làm nhiệm vụ truy xuất bộ nhớ dữ liệu để đọc hay ghi dữ liệu.

Khối DataGen làm nhiệm vụ chuyển dữ liệu đọc hay ghi thành dạng chuẩn (byte, haft-word, word, unsigned byte, unsigned haft-word) tương ứng với từng câu lệnh load/store.

Bộ mux trên sơ đồ cấu trúc tầng Mem được dùng để sửa các lỗi Data Hazard ở tầng Mem. Xem mục 5. FORWARDING UNIT

Hoạt động cơ bản của IL1 Cache:

Địa chỉ từ alu_out_mem sẽ được đưa đến L1 Data Cache và Data Victim Cache.

Nếu có yêu cầu đọc dữ liệu từ bộ nhớ, DATA CACHE CONTROLLER sẽ kiểm tra xem L1 Data Cache và Data Victim Cache có chứa dữ liệu cần đọc không:

- Có: DATA CACHE CONTROLLER kéo DCC_halt xuống mức thấp và xuất dữ liệu đọc được qua raw_data.
- Không: DATA CACHE CONTROLLER tích cực DCC_halt để yêu cầu dừng pipeline chờ dữ liệu được lấy từ các tầng memory chậm hơn (L2 Cache, Main Memory, …)

Nếu có yêu cầu ghi dữ liệu vào bộ nhớ, DATA CACHE CONTROLLER sẽ kiểm tra xem L1 Data Cache và Data Victim Cache có chứa ô nhớ cần ghi không:

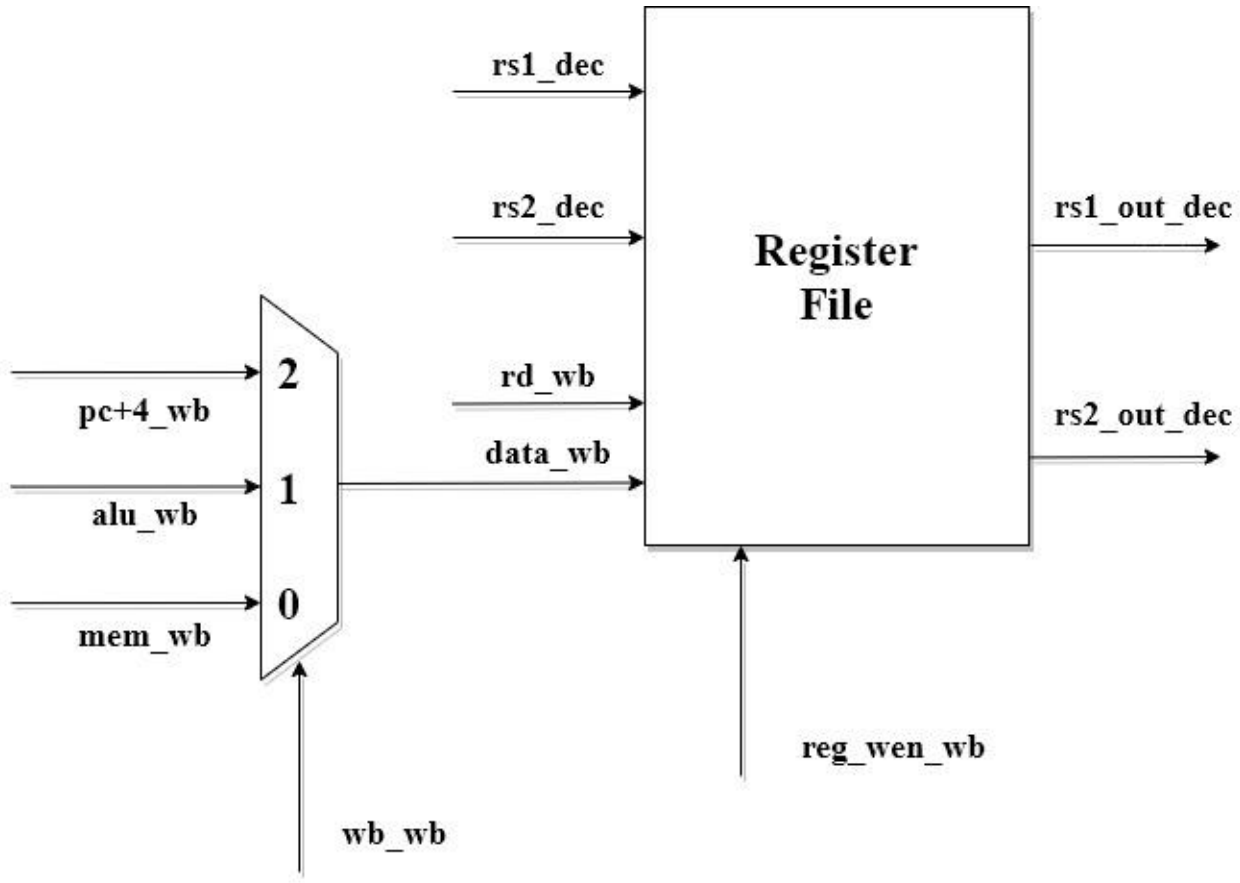
- Có: ghi đè dữ liệu vào ô nhớ tương ứng
- Không: đưa dữ liệu vào Write Buffer

Xem mục 7. CACHE

Danh sách các tín hiệu cần được truyền sang tầng tiếp theo:

Tên tín hiệu	Mô tả
rd_addr_mem	Địa chỉ thanh ghi đích ở tầng MEM
alu_out_mem	Kết quả từ bộ ALU ở tầng Mem
mem_mem	Dữ liệu đọc ra từ bộ nhớ
pc4_mem	PC + 4 ở tầng Mem
decoder_out_mem	Tổ hợp tín hiệu control cần gửi đến tầng WB

2.5 WRITE BACK STAGE

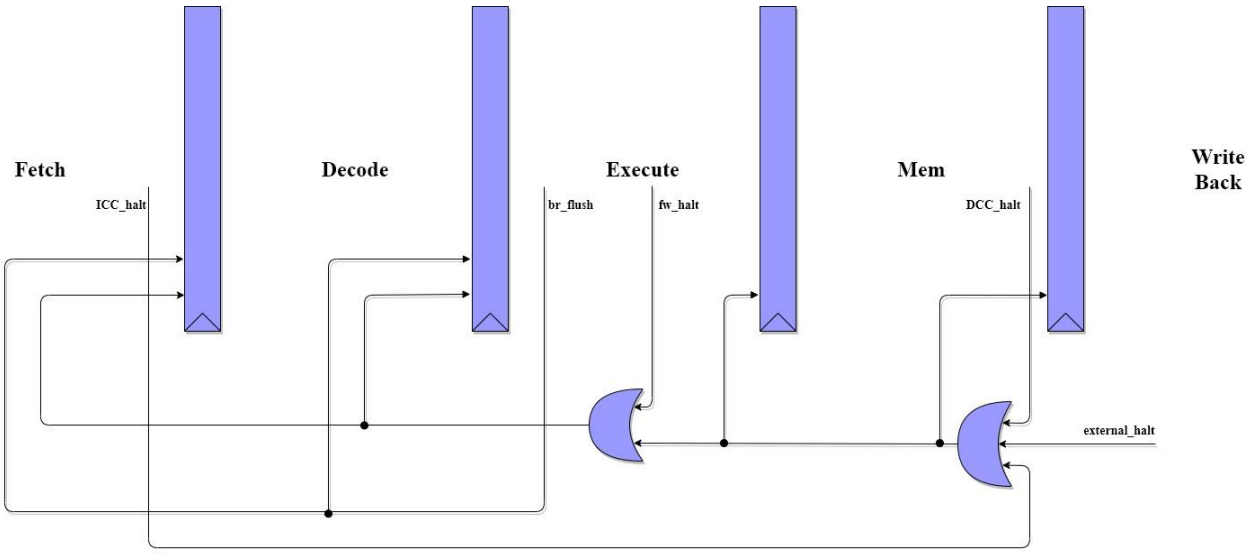


Hình 2.5: Write Back Stage

Tầng Write Back thực hiện việc ghi dữ liệu sau khi tính toán vào Register File.

Register File ở đây cũng chính là Register File ở tầng Decode. Để giải quyết lỗi Hardware Hazard, Register File sẽ hiện thực việc cập nhập dữ liệu theo cạnh lên và xuất dữ liệu ở cạnh xuống.

3 PIPELINE REGISTER



Hình 3: Pipeline Register

Các tín hiệu halt từ tầng sau sẽ dừng các tầng trước nó

- DCC_halt: tín hiệu từ Data Cache Controller yêu cầu dừng toàn bộ Pipeline.
- fw_halt: tín hiệu từ bộ Forwarding Unit yêu cầu dừng các tầng Fetch-Decode-Execute-Mem từ
- ICC_halt: tín hiệu từ Instruction Cache Controller yêu cầu dừng toàn bộ Pipeline.
- external_halt: tín hiệu từ bên ngoài lõi RVS192 yêu cầu dừng toàn bộ Pipeline. Đây là tín hiệu được thêm vào cho các ứng dụng sau này, ở đồ án 192 này tín hiệu này sẽ được nối đất.

Tín hiệu br_flush từ bộ Branch Handling Unit sẽ xóa các câu lệnh fetch sai ở các tầng fetch và decode.

4 CẤU HÌNH RVS192

4.1 CẤU HÌNH PHẦN CỨNG HỖ TRỢ

Macro	Mặc định	Mô tả
IL1 Cache		
INST_VICTIM_CACHE	Có	Tích hợp Victim Cache
ICACHE_ALRU (1)	Có	Sử dụng thuật toán thay thế ~LRU
ICACHE_FIFO (1)	Không	Sử dụng thuật toán thay thế FIFO
DL1 Cache		
DATA_VICTIM_CACHE	Có	Tích hợp Victim Cache
DCACHE_ALRU (2)	Có	Sử dụng thuật toán thay thế ~LRU
DCACHE_FIFO (2)	Không	Sử dụng thuật toán thay thế FIFO
Branch Prediction Unit		
HYBRID_BP (3)	Có	Sử dụng bộ Hybrid Branch Prediction (Local + Gshare)
LOCAL_BP (3)	Không	Chỉ sử dụng bộ Local Branch Predictor

GSHARE_BP [(3)]	Không	Chỉ sử dụng bộ GShare Branch Predictor

Bảng 4.1: Bảng Macro cấu hình phần cứng hỗ trợ

Chú ý: ***Các phần cứng không có Macro cài đặt được mặc định là luôn có sẵn.***

*(1) (2) **Define 1 trong 2***

*(3) **Define 1 trong 3***

4.2 CẤU HÌNH THÔNG SỐ PHẦN CỨNG

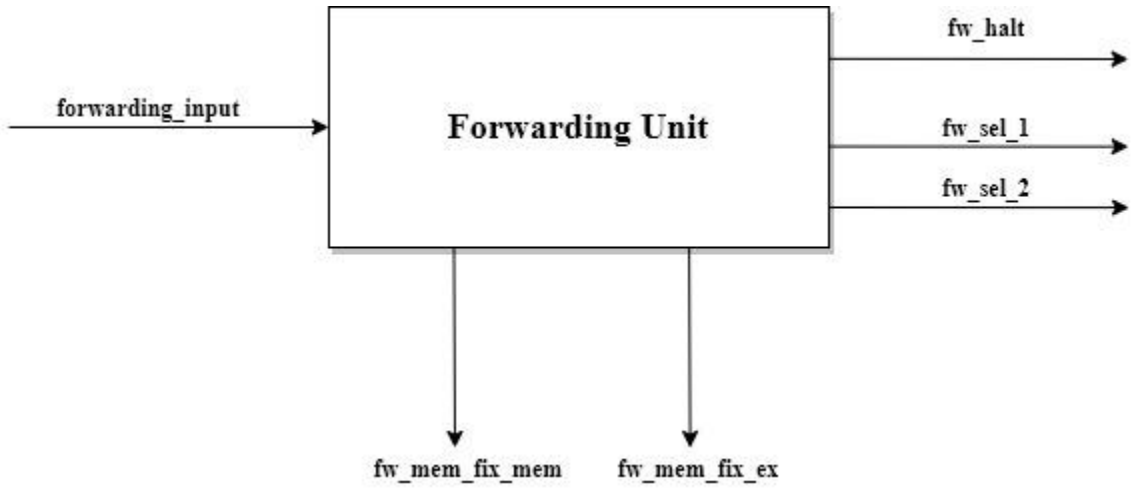
RVS192 được thiết kế theo cấu trúc RISC-V cpu 32-bit do đó parameter DATA_WIDTH được cố định là 32 và không được phép thay đổi bởi người dùng.

Parameter	Mặc định	Mô tả
IL1 Cache		
ICACHE_SIZE	32	Dung lượng của L1 Instruction Cache (Kbyte)
ICACHE_BLOCK_SIZE	64	Dung lượng của một block L1 Instruction Cache (byte)
ICACHE_WAY	4	Instruction Cache associativity
DL1 Cache		
DCACHE_SIZE	32	Dung lượng của L1 Data Cache (Kbyte)
DCACHE_BLOCK_SIZE	64	Dung lượng của một block L1 Instruction Cache (byte)
DCACHE_WAY	8	Data Cache associativity
DCACHE_WB_DEPTH	10	Dung lượng của Data Cache Write Buffer
L2 Cache		

L2_CACHE_SIZE	256	Dung lượng của L2 Cache (Kbyte)
L2_CACHE_BLOCK_SIZE	64	Dung lượng của một block L2 Cache (byte)
L2_CACHE_WAY	16	L2 Cache associativity
L2_CACHE_WB_DEPTH	20	Dung lượng của L2 Cache Buffer
Branch Prediction Unit		
GSHARE_HISTORY_LENGTH	12	Số câu lệnh rẻ nhánh được lưu lại ở bộ Gshare Branch Predictor
LOCAL_HISTORY_LENGTH	10	Số câu lệnh rẻ nhánh được lưu lại ở bộ Local Branch Predictor
GSHARE_GPT_DEPTH	4096	Số ô nhớ được sử dụng cho GPT của bộ Gshare Branch Predictor
LOCAL_LPT_DEPTH	1024	Số ô nhớ được sử dụng cho LPT của bộ Local Branch Predictor
LOCAL_LHT_DEPTH	1024	Số ô nhớ được sử dụng cho LHT của bộ Local Branch Predictor

Bảng 4.2: Bảng parameter cấu hình phần cứng

5 FORWARDING UNIT

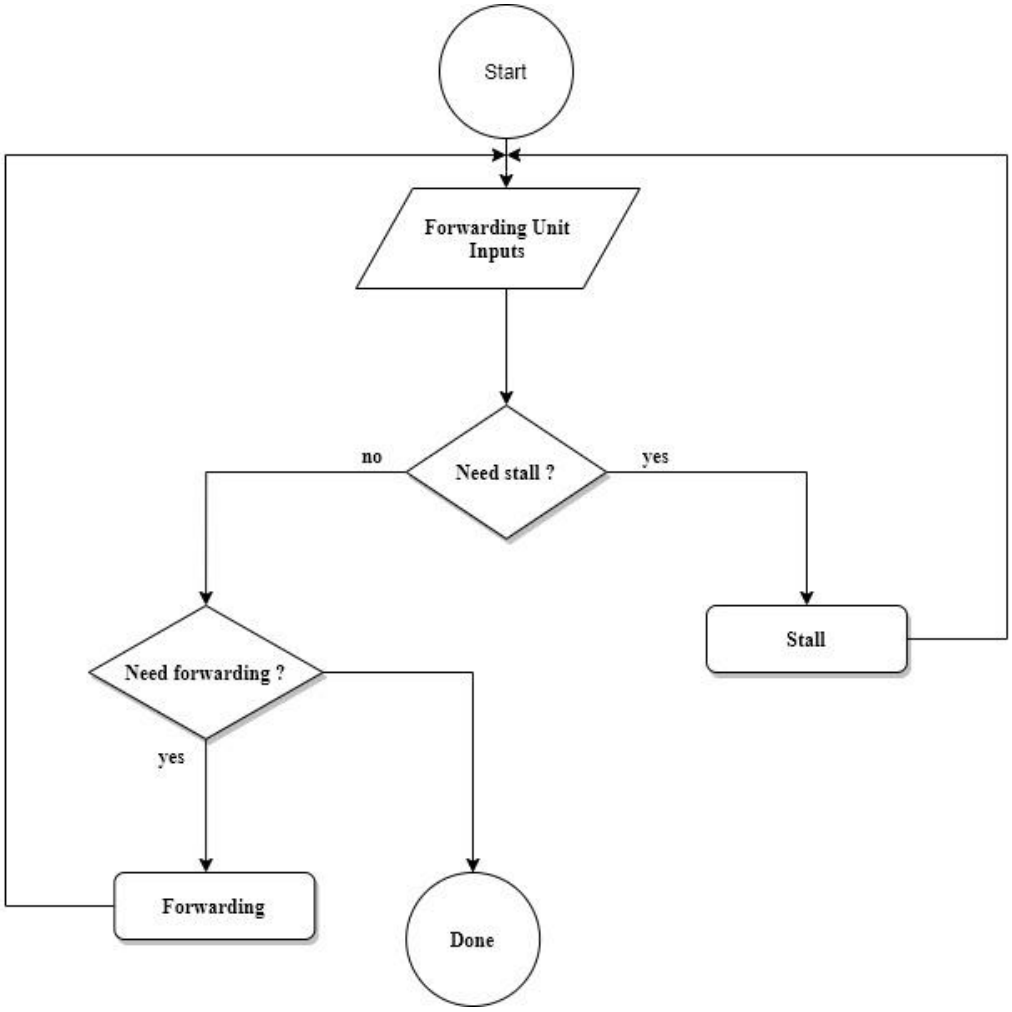


Hình 5.1: Forwarding Unit

Bộ Forwarding Unit làm nhiệm vụ sửa lỗi Data Hazard:

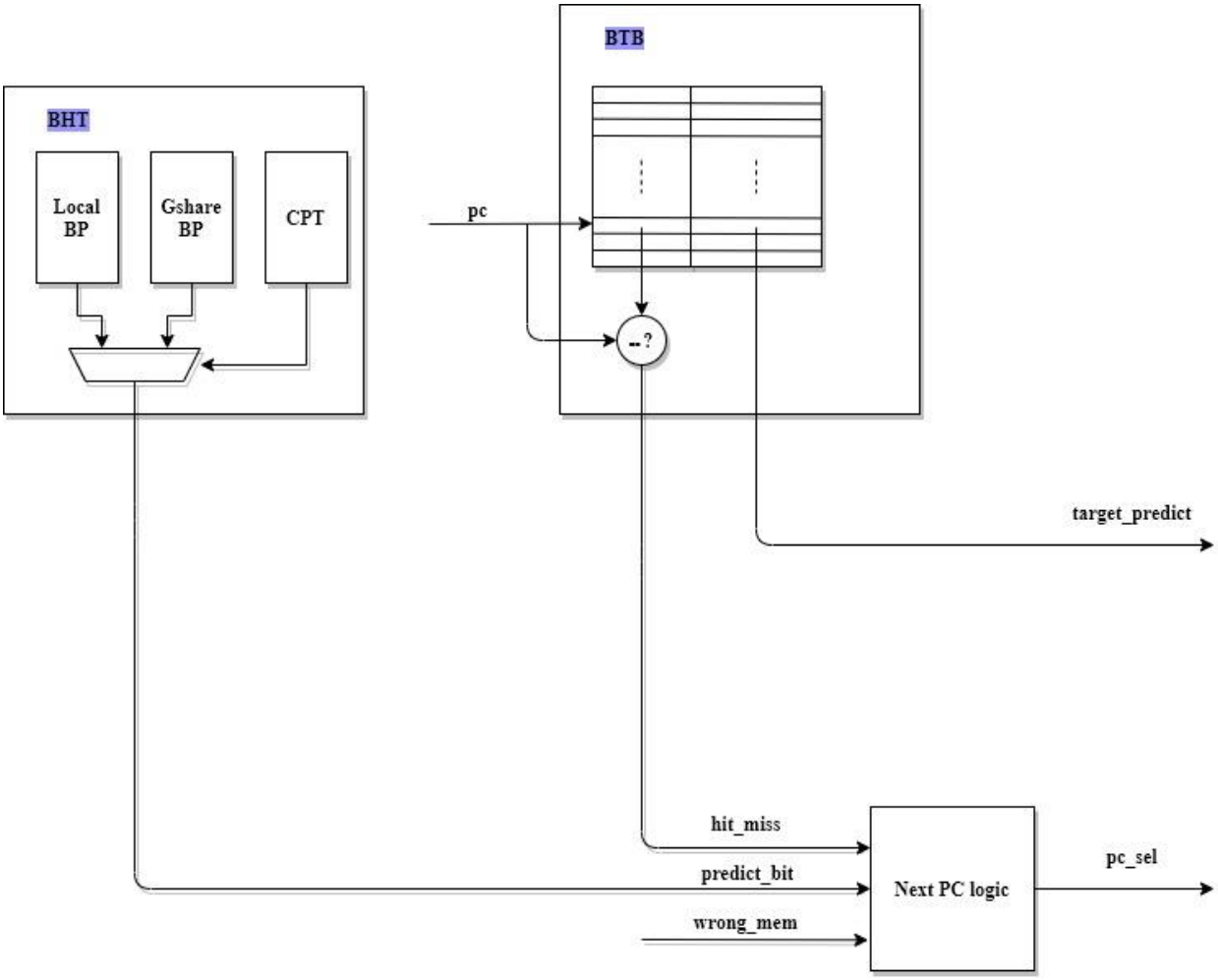
- Các lệnh chỉ sử dụng thanh ghi (R-Format): lỗi data hazard được giải quyết bằng cách forwarding kết quả đã hoàn thành ở các tầng sau để cung cấp cho tầng phía trước. Pipeline vẫn hoạt động xuyên suốt mà không cần phải dừng lại.
- Các lệnh truy cập data memory (một phần của I-Format và S-Format): các lỗi data hazards khi lưu dữ liệu vào memory cũng được giải quyết bằng forwarding. Tuy nhiên lỗi use-after-load (lệnh theo sau yêu cầu sử dụng thanh ghi có dữ liệu được lấy ra từ ô nhớ của lệnh ngay trước nó). Lỗi này không thể giải quyết được bằng forwarding do đó Pipeline sẽ bị dừng một chu kỳ để đảm bảo use-after load được sửa lỗi.

Dưới đây là thuật toán của bộ Forwarding Unit:



Hình 5.2: Forwarding Unit Algorithm

6 BRANCH PREDICTION UNIT



Hình 6.0: Branch Prediction Unit sử dụng Hybrid BP

Bộ Branch Prediction Unit (BPU) làm nhiệm vụ đoán lệnh nhảy có được taken hay không, giúp tăng CPI của lõi RVS192.

Bộ BPU gồm 3 khối chính được thể hiện trên Hình 6.1:

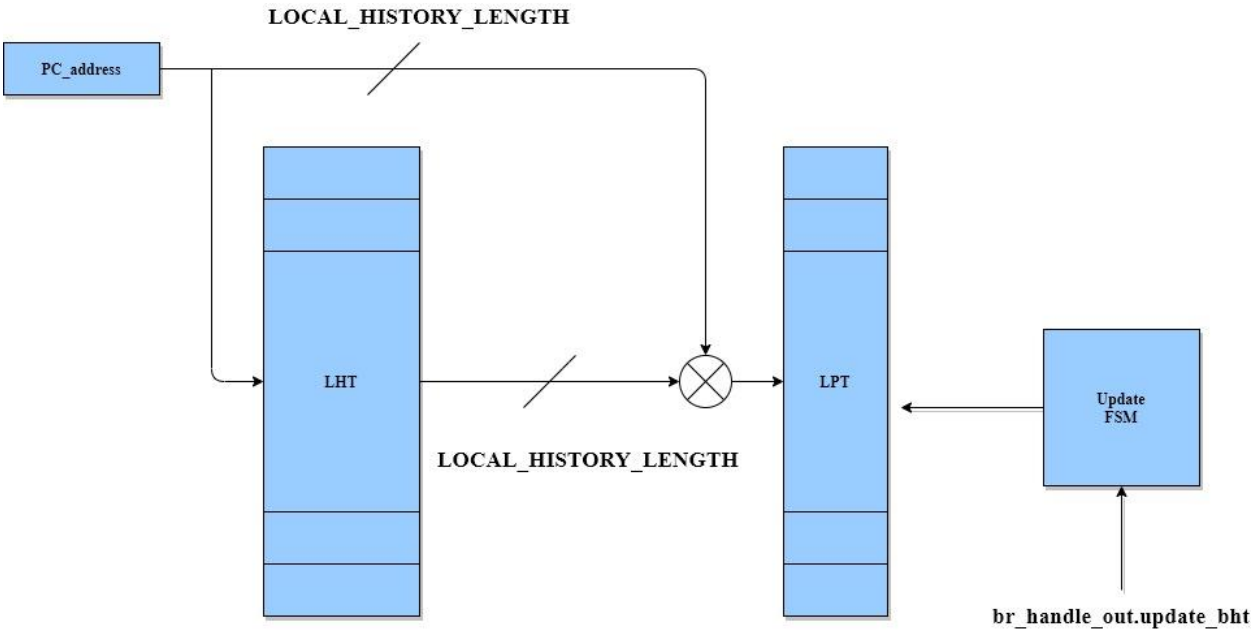
- BHT: Dựa vào kết quả dự đoán của bộ Prediction (chọn 1 trong 3 cấu hình: Local BP, Gshare BP, Hybrid BP), khối BHT sẽ xuất tín hiệu tiên đoán predict_bit gửi đến bộ Next PC logic

- BTB: Chứa các giá trị Tag và địa chỉ nhảy tới tương ứng từng câu lệnh nhảy. Khi BTB phát hiện ra địa chỉ nhảy tới của lệnh nhảy hiện tại , nó sẽ tích cực tín hiệu hit_miss, và xuất địa chỉ nhảy tới tương ứng ra tín hiệu target_predict.
- Next PC logic: căn cứ vào dự đoán của BHT, địa chỉ lệnh được lưu trong BTB và tín hiệu sửa lỗi wrong_mem được gửi đến từ tầng Mem (tín hiệu br_handle_out .wrong_ex được chuyển sang tầng Mem để đảm bảo các tín hiệu được lấy mẫu đúng định thì), khối Next PC logic sẽ chọn địa chỉ lệnh chính xác cần được fetch.

Mặc định nếu người dùng không cài đặt lại, bộ BHT sẽ được cấu hình Hybrid BP với các thông số như sau:

- 1Kbyte PHT sử dụng bộ 2-bit Saturating Up/Down Counter Predictor
- 1Kbyte LHT với 10 bit lịch sử lệnh nhảy
- 4Kbyte GPT sử dụng bộ 2-bit Saturating Up/Down Counter Predictor
- 4Kbyte CPT sử dụng bộ 2-bit Saturating Up/Down Counter Predictor
- GBHR: thanh ghi 12-bit lưu lịch sử lệnh nhảy

6.1 LOCAL BRANCH PREDICTION



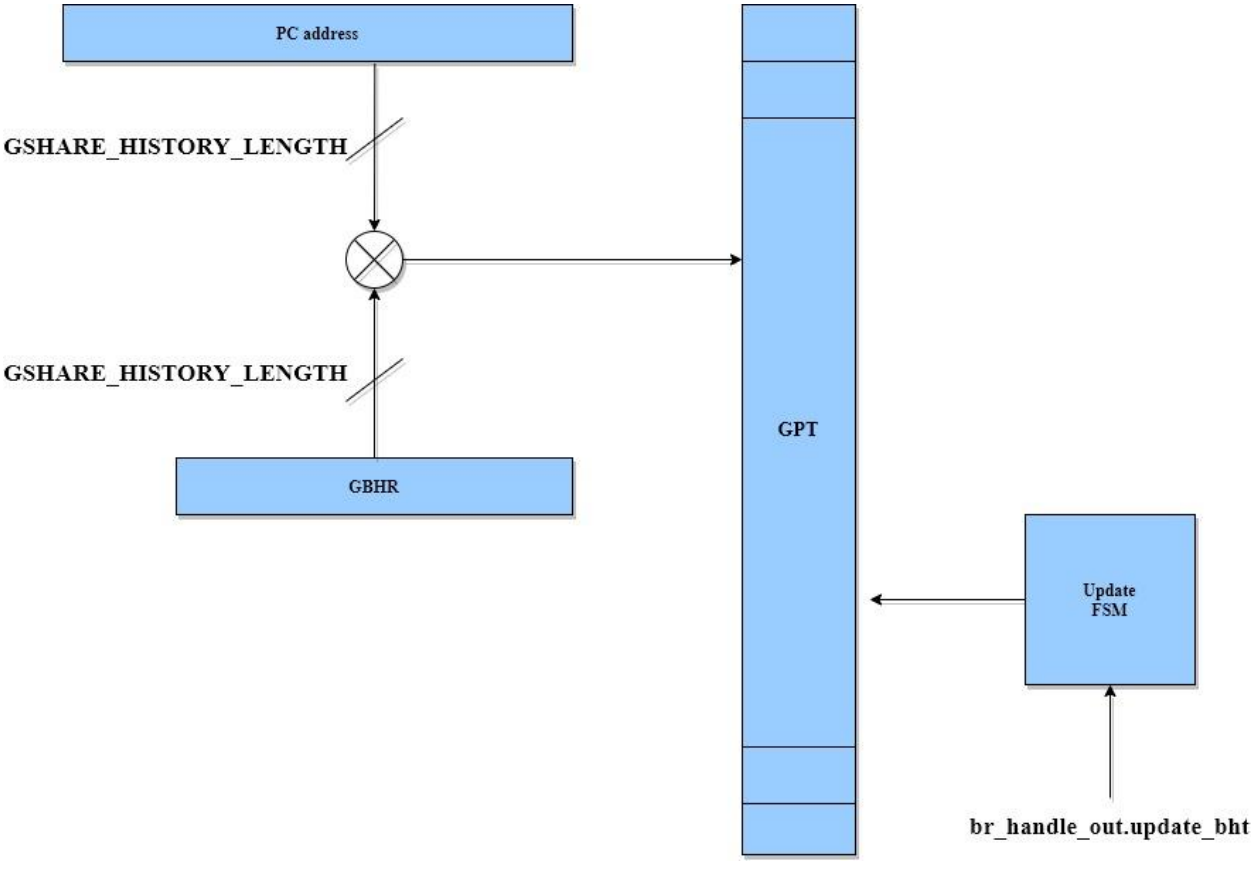
Hình 6.2: BHT với cấu hình Local BP

Bộ Local BP sử dụng lịch sử nhảy của từng lệnh nhảy để dự đoán xem lần tiếp theo gặp lại lệnh nhảy đó có nên taken hay không.

Mỗi ô nhớ LHT (Local History Table) đóng vai trò giống như 1 thanh ghi chứa lịch sử nhảy của từng lệnh nhảy. Khi gặp lại lệnh nhảy đó, LHT sẽ xuất ra các history pattern của đúng lệnh đó nhằm chọn địa chỉ ô nhớ dự đoán trong LPT. Trước khi được sử dụng làm địa chỉ trỏ đến LPT, các history pattern này sẽ được xor với pc lệnh nhảy để điều chế ra địa chỉ tương ứng. Cách làm này giúp giảm khả năng trùng lặp địa chỉ các các pattern thuộc các lệnh nhảy khác nhau.

Mỗi ô nhớ LPT (Local Pattern Table) chứa 2 bit predict được tạo ra từ bộ 2-level predictor tích hợp trong khối Update FSM. Bit cao của ô nhớ LPT tương ứng với lệnh nhảy sẽ được sử dụng làm predicted_bit.

6.2 GSHARE BRANCH PREDICTION



Hình 6.1: BHT với cấu hình Gshare BP

Việc sử dụng bộ Local BP để dự đoán dựa trên lịch sử nhảy của từng lệnh như vậy đôi khi sẽ cho ra xác suất dự đoán chính xác không tốt do các câu lệnh nhảy thường có sự liên quan đến nhau:

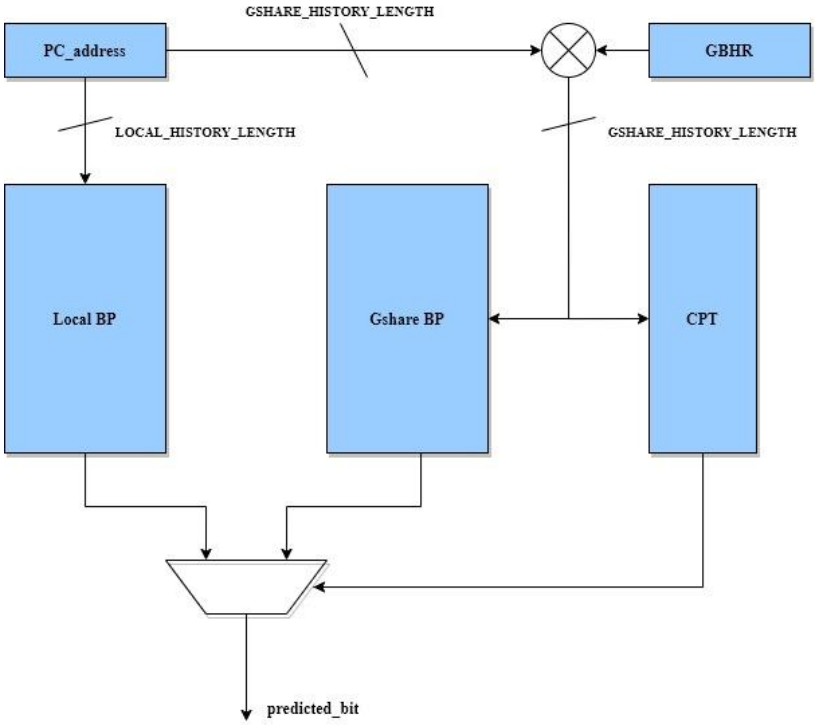
Vd:

```
…       if(a)
                b = 1;
…
        if(b)
                …
```

Để khắc phục vấn đề trên t có thể sử dụng bộ Gshare BP. Gshare BP sẽ cho ra hiệu suất cao hơn Local BP với số lượng tài nguyên tiêu tốn là ít hơn rất nhiều. Khác với Local BP lưu lịch sử nhảy của từng câu lệnh nhảy, Gshare BP chỉ sử dụng một thanh ghi GBHR (Global Branch History Register) để lưu lịch sử nhảy của tất cả các lệnh nhảy được thực hiện.

Mỗi ô nhớ GPT (Global Pattern Table) chứa 2 bit predict được tạo ra từ bộ 2-level predictor tích hợp trong khối Update FSM. Bit cao của ô nhớ GPT tương ứng với lệnh nhảy sẽ được sử dụng làm predicted_bit. Địa chỉ trỏ đến của các ô nhớ GPT được điều chế bằng phép xor giữa từng bit của thanh ghi GBHR và địa chỉ pc tiếp theo ở tầng Fetch (pc_in). Cách điều chế này giúp giảm xác suất trùng địa chỉ của các history pattern.

6.3 HYBRID BRANCH PREDICTION



Hình 6.3: BHT với cấu hình Hybrid BP

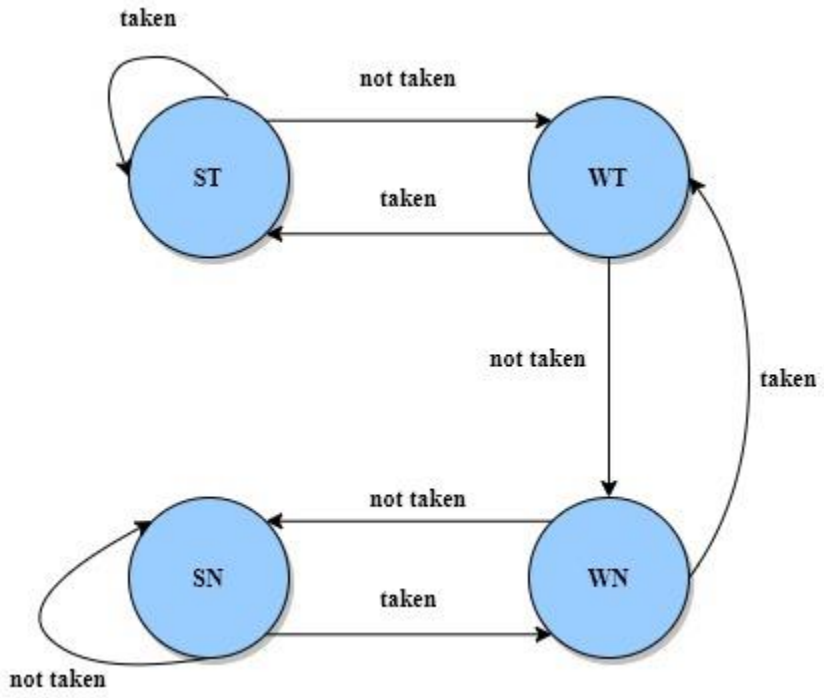
Để tối đa xác suất đoán đúng của bộ BP, người dùng sử dụng cấu hình Hybrid BP: đây là một bộ BP sử dụng song song Local BP và Gshare BP. Khi gặp một lệnh nhảy cả Local BP và Gshare BP đều cho ra dự đoán của riêng mình, bit cao của ô nhớ tương ứng với lệnh nhảy trong CPT sẽ dự đoán với lần gặp lệnh nhảy đó thì nên chọn kết quả dự đoán từ Local BP hay Gshare BP.

CPT (Choose Pattern Table) có cách hoạt động, lấy địa chỉ y hệt như GPT do đó thanh ghi GBHR của Gshare BP sẽ được đưa ra ngoài để dùng chung cho cả CPT và GPT. Mỗi ô nhớ GPT (Global Pattern Table) cũng chứa 2 bit predict được tạo ra từ bộ 2-level predictor tích hợp trong khối Update FSM. Khác với GPT và LPT dự đoán lệnh nhảy có nên taken hay không, CPT dự đoán nên chọn predict_bit từ Gshare BP hay Local BP để đạt được kết quả chính xác nhất.

6.4 2-LEVEL PREDICTOR

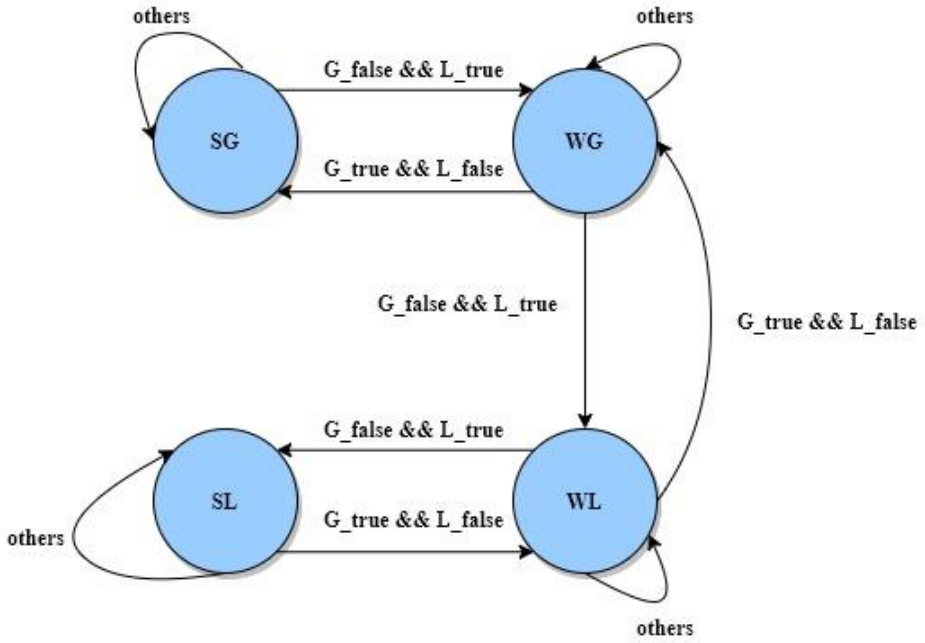
Bộ 2-Level Predictor thực chất là một FSM 4 trạng thái.

- Đối vơi LPT và GPT: ST (Strongly Taken: 2’b11), WT (Weakly Taken: 2’b10), WN (Weakly Not Taken: 2’01) và SN (Strongly Not Taken: 2’b00). MSB của mỗi trạng thái chính là predict_bit.



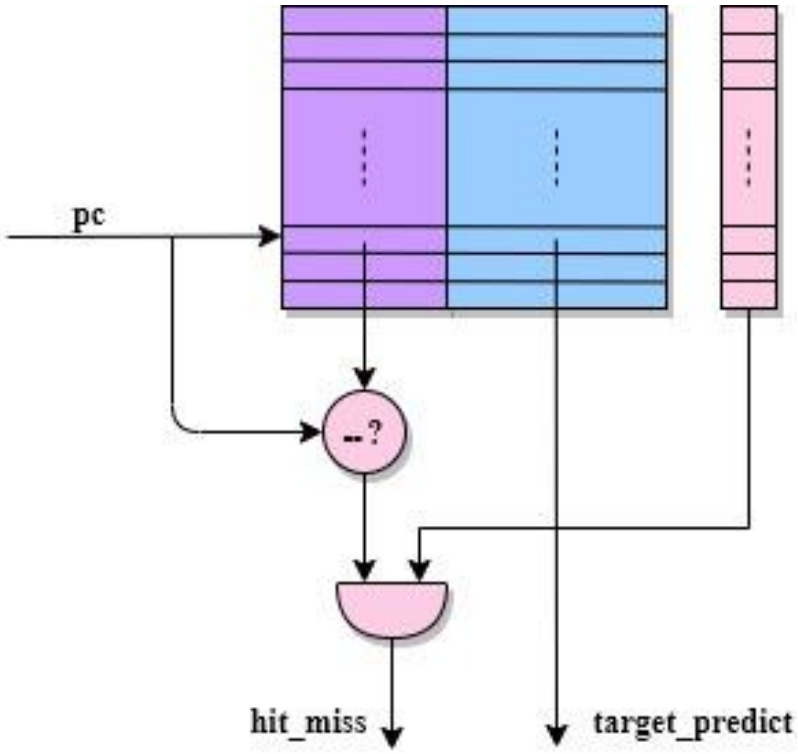
Hình 6.4.1: Sơ đồ chuyển trạng thái 2-bit Predictor của LPT và GPT

- Đối với CPT: SG (Strongly Gshare: 2'b11), WG (Weakly Gshare: 2'b10), WL (Weakly Local: 2'b01) và SL (Strongly Local: 2'b00). MSB = 1 sẽ chọn predict_bit từ bộ Gshare BP và MSB = 0 sẽ chọn predict_bit từ bộ Local BP làm kết quả.



Hình 6.4.1: Sơ đồ chuyển trạng thái 2-bit Predictor của CPT

6.5 BRANCH TARGET BUFFER



Hình 6.5: BTB

BTB (Branch Target Buffer) thực chất là một bộ nhớ lưu trữ thông tin về các địa chỉ lệnh nhảy và địa chỉ nhảy tới của lệnh nhảy đó. BTB chứa 3 bộ nhớ chính: một bộ nhớ chứa phần Tag của pc lệnh nhảy, một bộ nhớ chứa pc nhảy tới của lệnh nhảy tương ứng và bộ nhớ cuối cùng chứa valid_bit cho biết câu lệnh nhảy hiện tại đã từng gặp hay chưa. Việc sử dụng thêm bộ nhớ valid_bit giúp BTB tránh trường hợp sai sót ví dụ như: địa chỉ 0x000000 không chứa lệnh nhảy tuy nhiên trong bộ nhớ Tag lại lưu Tag của địa chỉ này…

Khi BTB phát hiện ra lệnh nhảy hiện hành đã được lưu trong bộ nhớ, BTB sẽ tích cực tín hiệu hit_miss và đưa địa chỉ nhảy tới ra đường target_predict.

7 CACHE

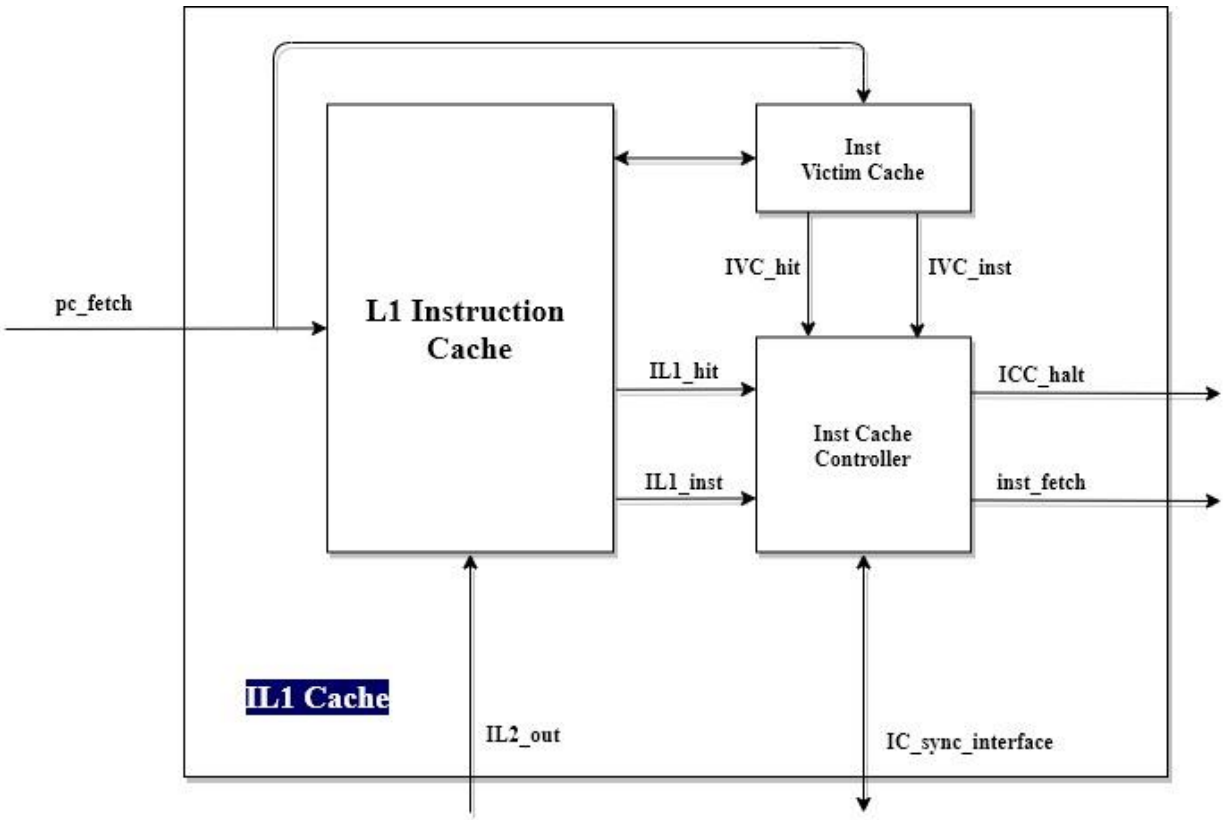
Phần này tập trung mô tả chức năng và cách thức hoạt động của Cache chứ không đi sâu vào chi tiết cấu trúc bên trong Cache. Tùy theo cách cài đặt, cấu trúc bên trong của Cache sẽ sinh ra các mạch khác nhau, tuy nhiên chức năng hoạt động vẫn tương tự. Dưới đây là bảng mô tả các thông số và cấu hình của Cache. Các phần đề mục bên dưới mô tả hoạt động của Cache khi được cấu hình đầy đủ chức năng

Name	**Default**	**Configurability**
IL1 CACHE & DL1 CACHE		
Organization	Split $I & $D	unable
Size	$I: 32Kb, 64b/block $D: 32Kb, 64b/block	able
Associativity	$I: 4-way $D: 8-way	able
Updata Policy	ALRU, Critical word first	unable
Data Cache Write Policy	Write Buffer, Write Allocate	unable
L2 CACHE		
Organization	Share Cache	unable
Size	256Kb, 64b/block	able
Associativity	16-way	able

Updata Policy	ALRU, Critical word first	unable
Write Policy	Write Buffer, Write Allocate	unable

Bảng 7.0: Bảng mô tả cấu hình Cache của RVS192

7.1 IL1 CACHE



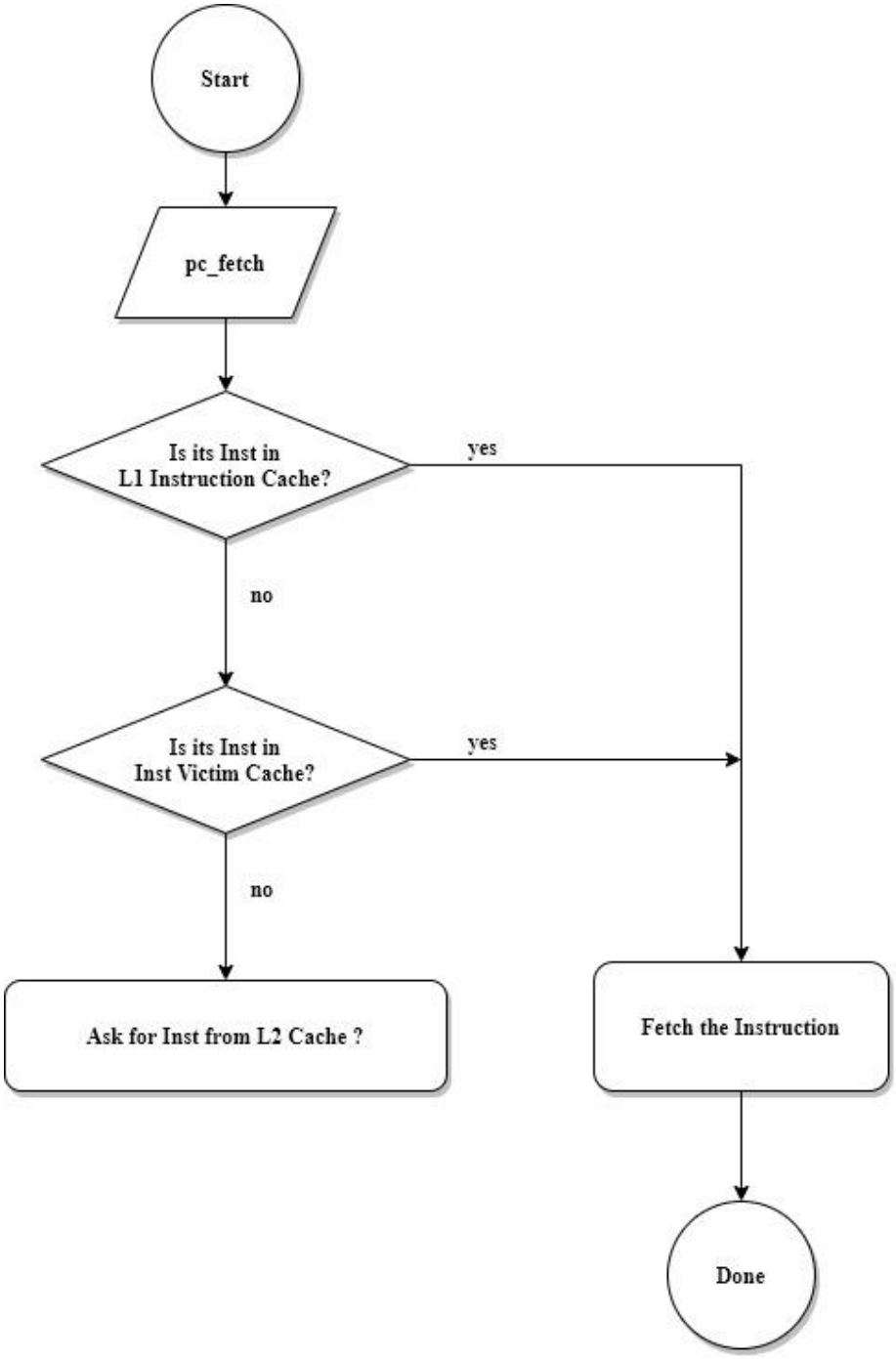
Hình 7.1.1: IL1 Cache

Như hình 7.1.1, ta có thể thấy IL1 Cache chứa 3 khối chính: L1 Instruction Cache, Inst Victim Cache và Inst Cache Controller. Trong đó:

- L1 Instruction Cache: là bộ nhớ lệnh chính của IL1 Cache. Khi phát hiện lệnh tương ứng với pc hiện thời có trong bộ nhớ, L1 Instruction Cache sẽ tích cực tín hiệu IL1_hit và gửi câu lệnh tương ứng trên đường IL1_inst đưa vào Inst Cache Controller.

- Inst Victim Cache: là bộ nhớ lệnh phụ của IL1 Cache, nơi đây chứa các câu lệnh bị đẩy ra khỏi IL1 Cache. Khi phát hiện lệnh tương ứng với pc hiện thời có trong bộ nhớ, Inst Victim Cache sẽ tích cực tín hiệu IVC_hit và gửi câu lệnh tương ứng trên đường IVC_inst đưa vào Inst Cache Controller
- Inst Cache Controller: là một FSM giúp điều khiển hoạt động của IL1 Cache. Khi có miss xảy ra Inst Cache Controller sẽ tích cực tín hiệu ICC_halt yêu cầu dừng Pipeline đến khi miss đó kết thúc.

Giải thuật hoạt động của IL1 Cache sẽ được thể hiện trong hình bên dưới:

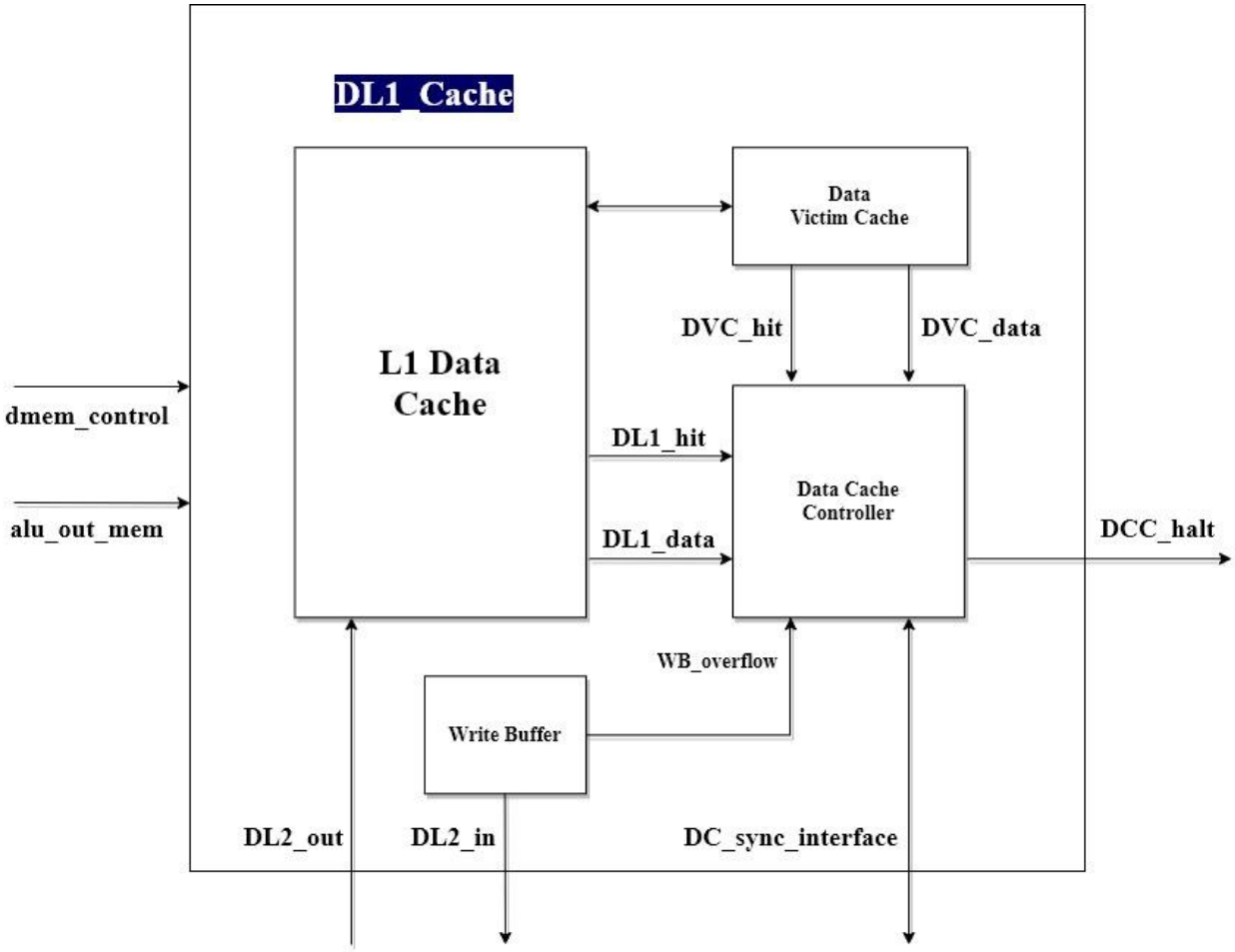


Hình 7.1.2: Inst Cache Controller Algorithm

Khi có miss xảy ra (cả IL1 Instruction Cache và Inst Victim Cache đều không có lệnh cần fetch), Inst Cache Controller sẽ gửi tín hiệu yêu cầu L2 Cache gửi block chứa dữ liệu mong muốn đến IL1 Instruction Cache, đồng thời tích cực ICC_halt yêu cầu dừng Pipeline. Block dữ liệu được thay thế sẽ được đưa vào Inst Victim Cache. Sau một vài

chu kỳ clock, block chứa dữ liệu mong muốn sẽ được ghi vào IL1 Instruction Cache. Do thiết kế Cache này hỗ trợ Critical word first, nên lệnh mong muốn bị miss ban đầu sẽ được đưa đến trước để cho phép Pipeline hoạt động, các câu lệnh còn lại trong block dữ liệu sẽ được đưa vào IL1 Instruction Cache song song với hoạt động của Pipeline. Nếu trong thời điểm IL1 Instruction Cache còn đang được update dữ liệu mà có một miss mới, Inst Cache Controller sẽ tích cực ICC_halt trở lại và chờ cho block dữ liệu cũ được update hoàn toàn và tiếp túc quá trình update dữ liệu như trên.

7.2 DL1 CACHE



Hình 7.2.1: DL1 Cache

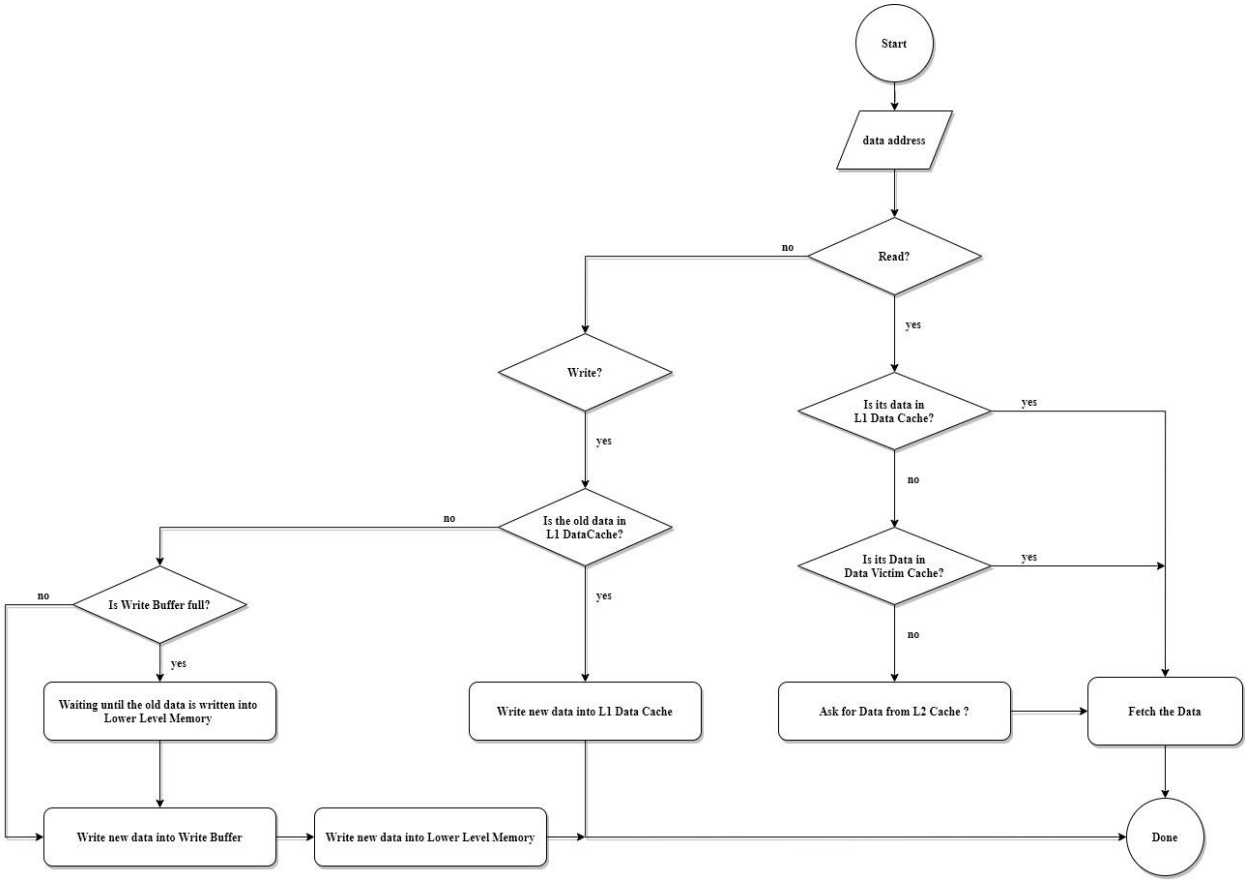
Như hình 7.1.1, ta có thể thấy DL1 Cache chứa 4 khối chính: L1 Data Cache, Data Victim Cache, Write Buffer và Data Cache Controller. Trong đó:

- L1 Data Cache: là bộ nhớ dữ liệu chính của DL1 Cache. Khi phát hiện ô nhớ tương ứng với địa chỉ hiện thời có trong bộ nhớ. Đối với lệnh đọc, L1 Data

Cache sẽ tích cực tín hiệu DL1_hit và gửi câu lệnh tương ứng trên đường DL1_data đưa vào Data Cache Controller. Đối với lệnh ghi, L1 Data Cache cũng sẽ tích cực tín hiệu DL1_hit và ghi dữ liệu vào ô nhớ mong muốn.

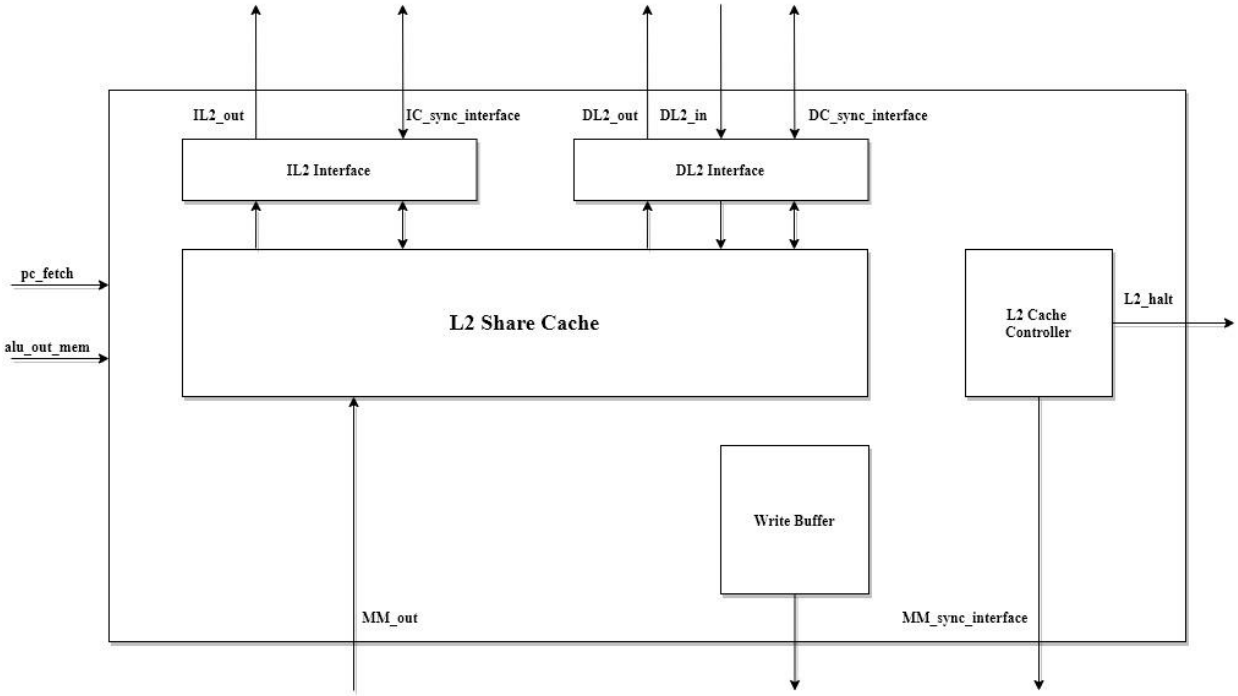
- Data Victim Cache: là bộ nhớ dữ liệu phụ của DL1 Cache, nơi đây chứa các câu lệnh bị đẩy ra khỏi DL1 Cache. Data Victim Cache chỉ được sử dụng cho lệnh đọc. Khi phát hiện lệnh tương ứng với pc hiện thời có trong bộ nhớ, Data Victim Cache sẽ tích cực tín hiệu DVC_hit và gửi câu lệnh tương ứng trên đường DVC_data đưa vào Data Cache Controller.
- Write Buffer: là bộ nhớ đệm chứa dữ liệu cần ghi. Write Buffer chỉ được sử dụng cho lệnh ghi. Nếu ô nhớ cần được ghi vào có trong L1 Data Cache, dữ liệu mới sẽ được ghi trực tiếp vào đó. Nếu không dữ liệu cần ghi sẽ được đưa vào Write Buffer. Pipeline vẫn được tiếp tục hoạt động. Sau một vài chu kỳ clock, dữ liệu trong Write Buffer sẽ được đưa vào các tầng Memory thấp hơn. Nếu như dữ liệu cũ trong Write Buffer chưa được đưa xuống tầng Memory thấp hơn mà đã có dữ liệu mới ghi đè vào vị trí dữ liệu chưa gửi đó, Data Cache Controller sẽ tích cực DCC_halt yêu cầu dừng Pipeline để hoàn thành việc ghi dữ liệu cũ. Sau đó DCC_halt được kéo xuống thấp và dữ liệu mới được ghi vào Write Buffer.
- Data Cache Controller là một FSM giúp điều khiển hoạt động của DL1 Cache. Khi có miss đọc xảy ra Data Cache Controller sẽ tích cực tín hiệu DCC_halt yêu cầu dừng Pipeline đến khi miss đó kết thúc và yêu cầu cập nhập dữ liệu vào DL1 Cache từ L2 Cache. Miss ghi chỉ tích cực tín hiệu DCC_halt khi Write Buffer đã đầy.

Giải thuật hoạt động của IL1 Cache sẽ được thể hiện trong hình bên dưới:



Hình 7.2: Data Cache Controller Algorithm

7.3 L2 CACHE



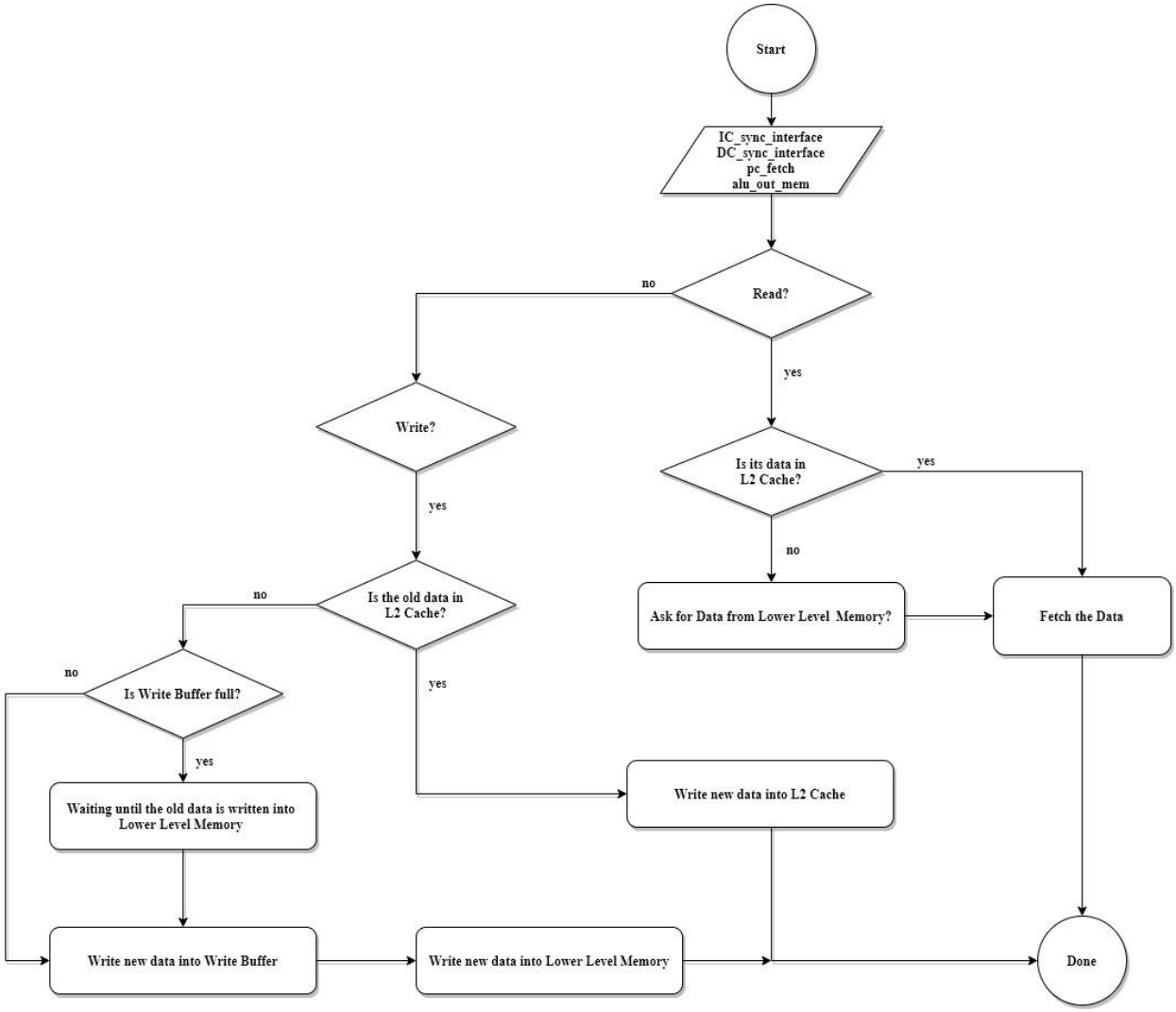
Hình 7.3.1: L2 Cache

Như hình 7.3.1, ta có thể thấy L2 Cache chứa 5 khối chính: IL2 Interface, DL2 Interface, L2 Cache Controller và Write Buffer. Trong đó:

- IL2 Interface: là phần logic giao tiếp với IL1 Cache, giúp đảm bảo dữ liệu được đồng bộ.
- DL2 Interface: là phần logic giao tiếp với DL1 Cache, giúp đảm bảo dữ liệu được đồng bộ.
- L2 Share Cache: là bộ nhớ dữ liệu dùng chung cho cả data và instruction. Khi phát hiện ô nhớ tương ứng với địa chỉ hiện thời có trong bộ nhớ. Đối với lệnh đọc, dữ liệu sẽ được đưa đến IL2 Interface hay DL2 Interface tùy vào dữ liệu cần là data hay instruction. Đồng thời tín hiệu L2_hit cũng sẽ được tích cực để báo hiệu dữ liệu hợp lệ. Đối với lệnh ghi, dữ liệu mới sẽ được ghi trực tiếp vào ô nhớ tương ứng với địa chỉ hiện thời.
- Write Buffer: Write Buffer: là bộ nhớ đệm chứa dữ liệu cần ghi. Write Buffer chỉ được sử dụng cho lệnh ghi. Nếu ô nhớ cần được ghi vào có trong L2 Cache, dữ liệu mới sẽ được ghi trực tiếp vào đó. Nếu không dữ liệu cần ghi sẽ được đưa vào Write Buffer. Pipeline vẫn được tiếp tục hoạt động. Sau một vài chu kỳ clock, dữ liệu trong Write Buffer sẽ được đưa vào các tầng Memory thấp hơn. Nếu như dữ liệu cũ trong Write Buffer chưa được đưa xuống tầng Memory thấp hơn mà đã có dữ liệu mới ghi đè vào vị trí dữ liệu chưa gửi đó,

L2 Cache Controller sẽ tích cực DCC_halt yêu cầu dừng Pipeline để hoàn thành việc ghi dữ liệu cũ. Sau đó DCC_halt được kéo xuống thấp và dữ liệu mới được ghi vào Write Buffer.

- L2 Cache Controller là một FSM giúp điều khiển hoạt động của L2 Cache. Khi có miss đọc xảy ra L2 Cache Controller sẽ yêu cầu cập nhập dữ liệu vào L2 Cache từ các tầng Memory thấp hơn. Miss ghi sẽ tích cực tín hiệu L2_halt khi Write Buffer đã đầy.



Hình 7.3.2: L2 Cache Algorithm