ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ

----------o0o----------

ĐỒ ÁN MÔN HỌC

RVS192 CPU

(Configurable 2 Level Cache, Branch Predictor;

5-Stage Pipeline RISC-V; RV32I)

GVHD: T.s Trần Hoàng Linh

SVTH: Lê Quang Hưng

MSSV: 1711631

TP. HỒ CHÍ MINH, THÁNG 7 NĂM 2020

LỜI CẢM ƠN

Trong thời gian làm đồ án môn học, em đã nhận được sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy cô, gia đình và bạn bè.

Em xin gửi lời cảm ơn chân thành đến T.s Trần Hoàng Linh, chủ nhiệm bộ môn Điện tử - Trường Đại học Bách Khoa Hồ Chí Minh; anh Trịnh Vũ Đăng Nguyên, giảng viên bộ môn Điện tử - Trường Đại học Bách Khoa Hồ Chí Minh; anh Cao Xuân Hải, kỹ sư tại Ampere Computing Vietnam đã tận tình hướng dẫn, chỉ bảo em trong suốt quá trình thực hiện đồ án môn học.

Em cũng xin chân thành cảm ơn các thầy cô giảng viên trong trường Đại Học Bách Khoa Hồ Chí đã dạy dỗ cho em kiến thức về các môn đại cương cũng như các môn chuyên ngành, anh Nguyễn Hùng Quân, kỹ sư tại Ampere Computing Vietnam đã chia sẽ các kiến thức về kỹ thuật thiết kế phần cứng, giúp em có được cơ sở lý thuyết vững vàng và tạo điều kiện giúp đỡ em trong suốt quá trình học tập.

Tp. Hồ Chí Minh, ngày 21 tháng 7 năm 2020.

Sinh viên

Lê Quang Hưng

TÓM TẮT ĐỒ ÁN

Đồ án này trình bày về cấu trúc lõi RVS192.

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.

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.

MỤC LỤC

DANH SÁCH HÌNH MINH HỌA

DANH SÁCH BẢNG SỐ LIỆU

1. GIỚI THIỆU

1.1 Tổng quan

Xu hướng tích hợp nhiều tác vụ trong một câu lệnh của cấu trúc tập lệnh CISC đã trở nên lỗi thời so với cấu trúc tập lệnh RISC.

Các thiết kế cấu trúc máy tính hiện đại thường được các công ty vi mạch giữ bí mật, các nhóm nghiên cứu phát triển CPU cũng thường được thỏa thuận không phát hành các tài liệu mô tả ưu điểm và hướng dẫn chi tiết về thiết kế của họ. Điều này dẫn đến sự thiếu hụt các thiết kế tham khảo hỗ trợ cho việc nghiên cứu và chỉ thường có sẵn trong các môi trường học thuật. Năm 2010, các chuyên gia của Đại học California, Berkeley đã tạo ra một ISA mã nguồn mở **RISC-V**, có thể sử dụng trong học thuật và bất kỳ thiết kế phần cứng hoặc phần mềm nào mà không yêu cầu tiền bản quyền.

Với nền tảng kiến thức từ môn học Cấu trúc máy tính của học kỳ trước, ở đồ án học kỳ 192 này, em đã tiếp tục phát triển và tối ưu kiến trúc lõi RISC-V cpu theo tập lệnh RV32I.

1.1 Nhiệm vụ đề tài

Nội dung 1: Nghiên cứu ngôn ngữ thiết kế phần cứng SystemVerilog

- Các hỗ trợ và cải tiến của SystemVerilog so với Verilog HDL.
- Cách sử dụng hỗ trợ của SystemVerilog để việc mô tả RTL code được tường minh và ít lỗi hơnVerilog HDL.

Nội dung 2: Nghiên cứu lý thuyết và các cấu hình hỗ trợ của bộ đệm Cache

- Cấu hình thông thường của Cache.
- Kỹ thuật nâng cao hit-rate.
- Kỹ thuật giảm miss-penalty.

Nội dung 3: Nghiên cứu và cải tiến cấu trúc của lõi RVS192

- Tối ưu phần cứng sử dụng.
- Giảm số chu kỳ chờ khi bộ Branch Predictor đoán sai lệnh nhảy.

2. LÝ THUYẾT

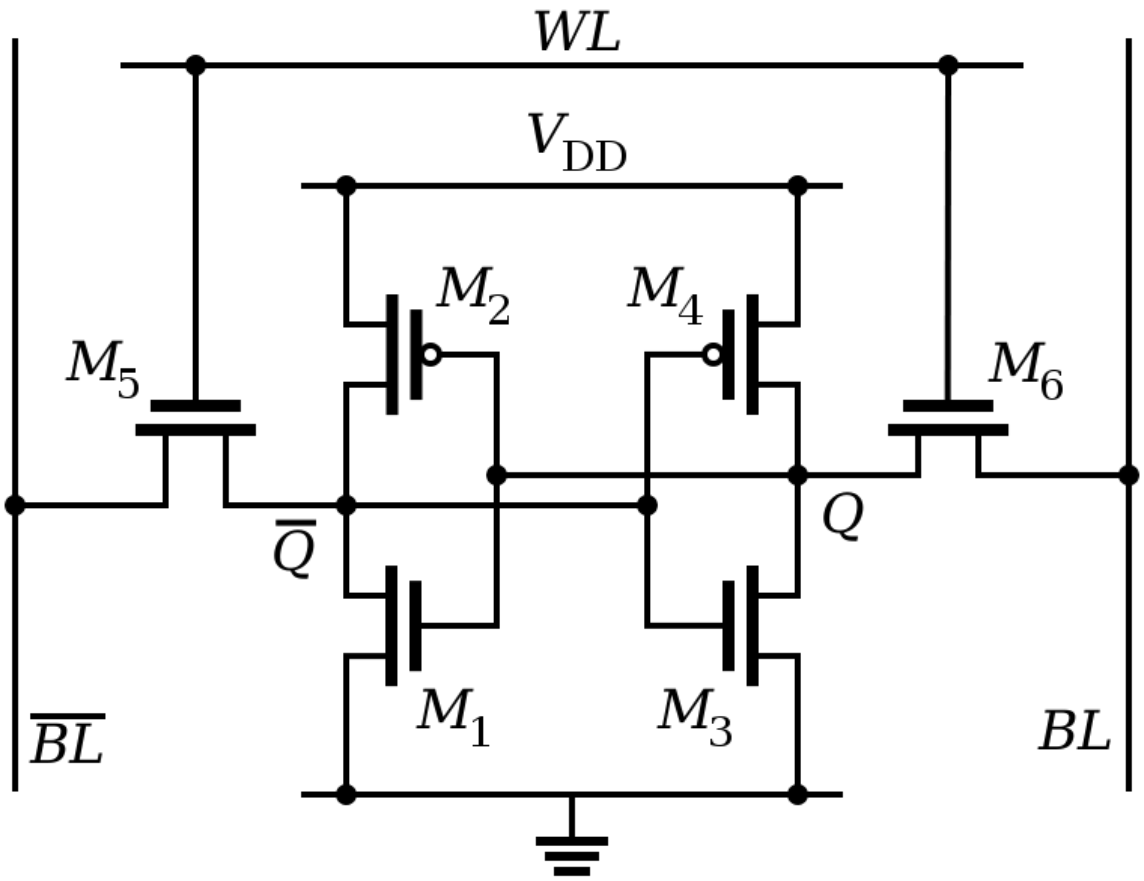
2.1 Các đặc tính của bộ nhớ lệnh và dữ liệu

- Temporal locality: Khi một lệnh hay dữ liệu được truy cập, có nhiều khả năng lệnh hay dữ liệu đó sẽ được sử dụng lại trong tương lai.
- Spatial Locality: Khi một lệnh hay dữ liệu được truy cập, nhiều khả năng các lệnh hay dữ liệu gần nó cũng sẽ được truy cập trong tương lai.

2.2 Memory Hierarchy

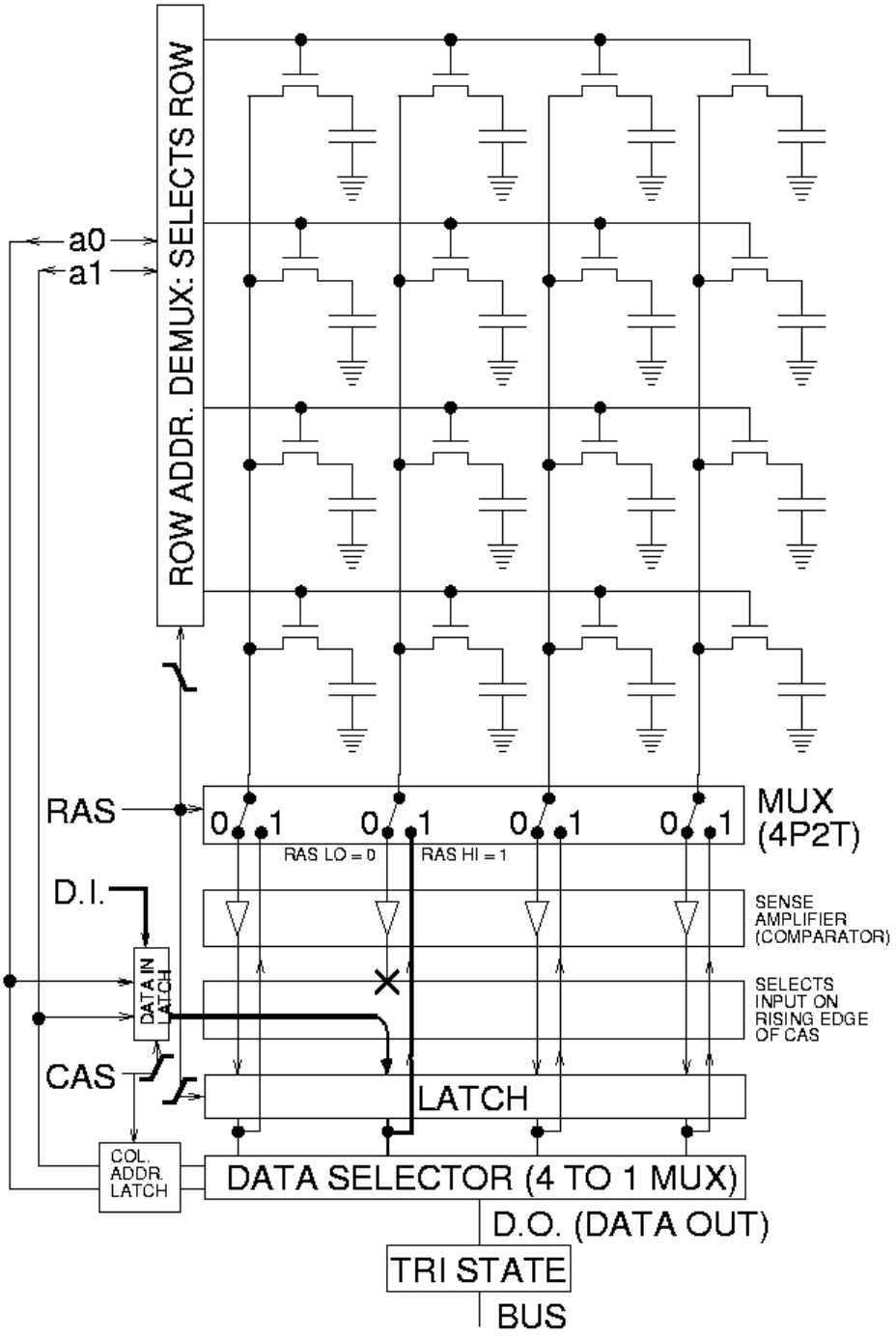
Các đặc tính của các loại bộ nhớ đều có các đặc điểm chung là: càng lớn thì càng chậm, càng nhỏ thì càng nhanh.

SRAM: sử dụng các cell Flip-flops để lưu trữ các bit dữ liệu, nhờ đó tốc độ truy xuất dữ liệu rất cao, tuy nhiên giá thành sản xuất của nó lại rất đắt. Do đó nó chỉ được sử dụng trong các ứng dụng yêu cầu tốc độ cao mà không cần nhiều bộ nhớ.



Hình 1: A six-transistor CMOS SRAM cell

DRAM: sử dụng tụ (capacitor) để lưu trữ dữ liệu, do đó DRAM cần phải làm mới dữ liệu cứ sau một khoảng thời gian được quy định. Với các đặc tính trên DRAM truy xuất dữ liệu với tốc độ chậm hơn so với SRAM, đồng thời cũng tốn năng lượng hơn, bù lại DRAM rẻ hơn rất nhiều và cũng ít chiếm diện tích hơn khi xét cùng một dung lượng lưu trữ.

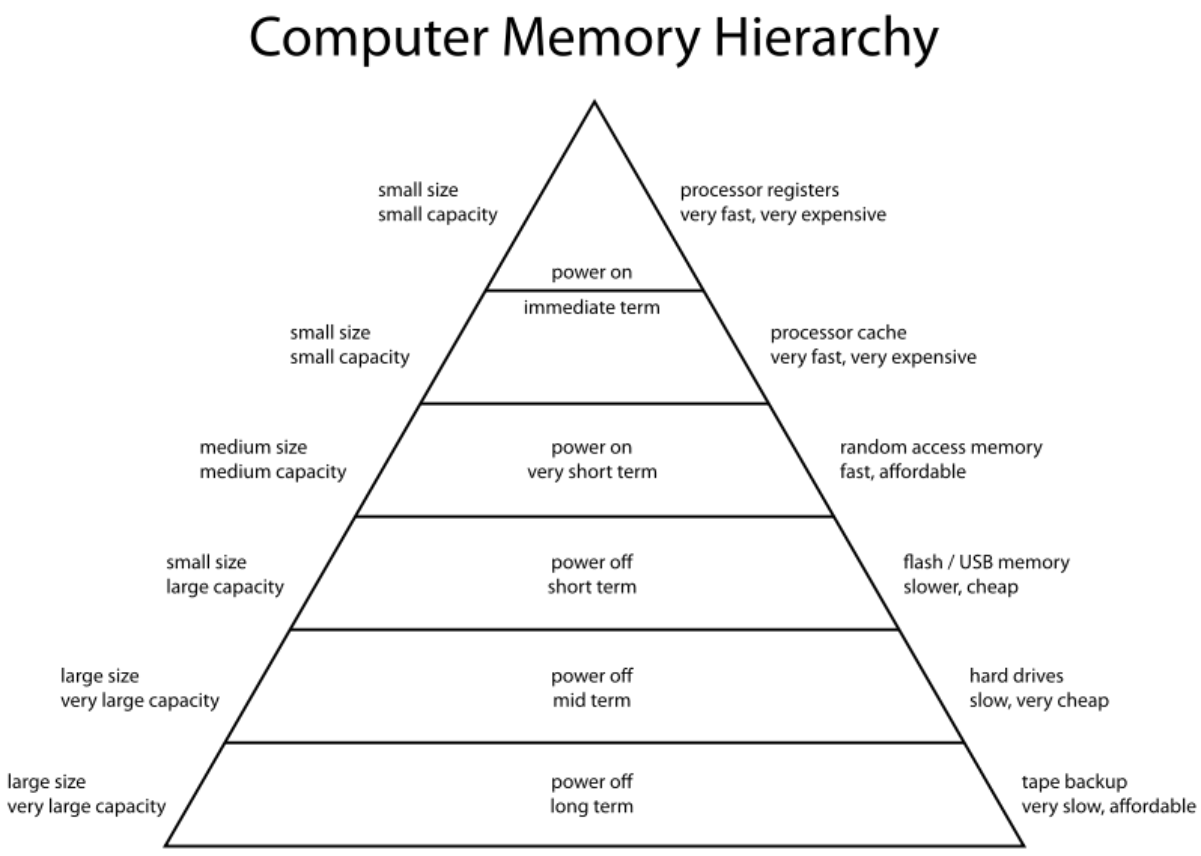


Hình 2: DRAM cell

Memory Hierarchy khai thác các đặc tính của bộ nhớ lệnh và dữ liệu bằng cách caching (giữ lệnh hay dữ liệu gần cpu) những câu lệnh hoặc dữ liệu có nhiều khả năng sẽ được sử dụng lại, đồng thời tận dụng các ưu điểm của từng loại bộ nhớ để tạo ra một hệ thống bộ nhớ truy xuất dữ liệu với tốc độ nhanh, dung lượng lớn và giá thành rẻ.

Một số từ khóa được sử dụng khi đề cập đến cache:

- Cache hit: dữ liệu cần truy cập được tìm thấy trong hệ thống cache.
- Cache miss: dữ liệu cần truy cập không được tìm thấy trong hệ thống cache.
- Hit time: thời gian tối thiểu để truy xuất dữ liệu khi có một cache hit xảy ra.
- Miss penalty: thời gian chờ để đưa dữ liệu từ các bộ nhớ có level thấp hơn lên level cao hơn, sau đó là đến cpu khi có một cache miss xảy ra.
- Hit-rate: tỷ lệ cache-hit so với tổng số truy cập vào bộ nhớ.
- Hit-rate: tỷ lệ cache-miss so với tổng số truy cập vào bộ nhớ (1 – hit-rate).



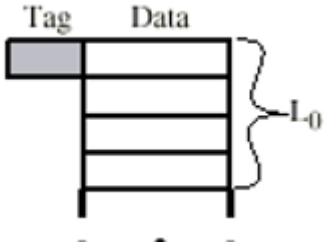
Hình 3: Memory Hierarchy

2.3 Cache Policy & Technique

2.3.1 Block Size

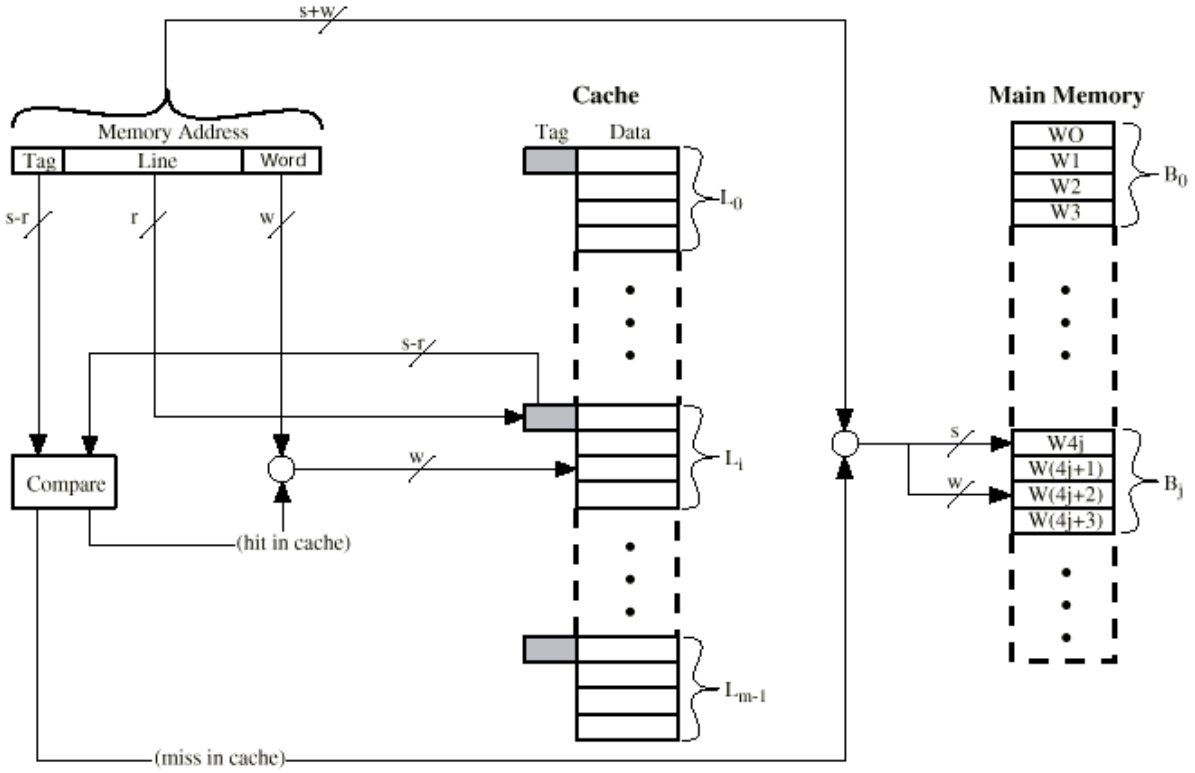
Theo Spatial Locality, khi một dữ liệu được sử dụng thì nhiều khả năng các dữ liệu gần đó cũng được sử dụng. Dựa vào nguyên tắc trên, một line dữ liệu thay vì chỉ chứa một word thì sẽ chứa một block gồm nhiều word dữ liệu, do đó khi có yêu cầu thay thế thì các word gần đó cũng sẽ được đưa đến Cache.

Tuy nhiên, cách làm này tuy tăng được hit rate tuy nhiên nó lại làm tăng miss penalty. Do đó, việc sử dụng Block Size lớn nên kết hợp với các kỹ thuật critical word first hoặc early restart.



Hình 4: Block Size

2.3.2 Direct Mapping Cache



Hình 5: Cấu trúc Direct Mapping Cache

Cấu hình Direct Mapping Cache là cấu hình đơn giản nhất của Dirrect Mapping Cache. Mỗi block dữ liệu của main memory sẽ được kết nối (map) vào duy nhất một cache line. Do đó, khả năng bị trùng lặp cache line là rất lớn, gây ra hiện tượng nút cổ chai (bottleneck).

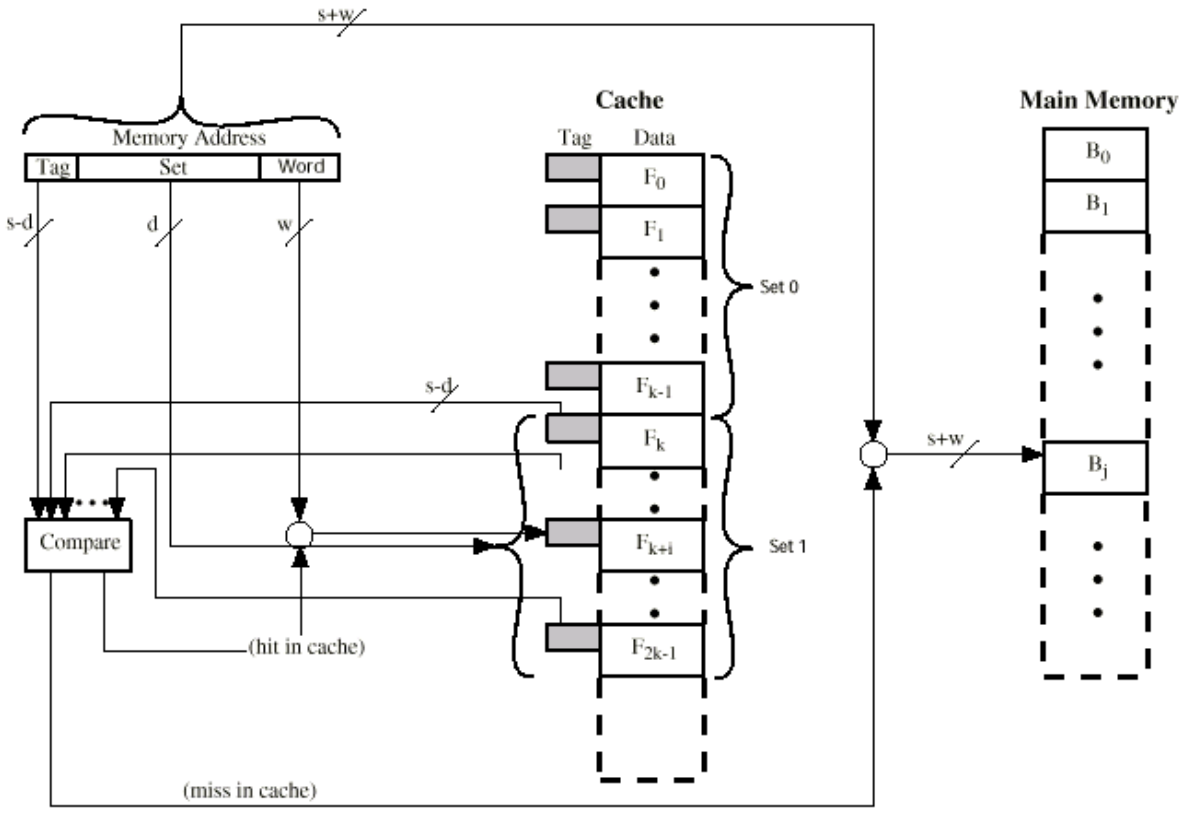
Ưu điểm:

- Đơn giản
- Rẻ

Nhược điểm:

- Nếu 2 block dữ liệu cùng được kết nối (map) vào một cache line được gọi liên tục, hiện tượng bottleneck xảy ra làm cho miss-rate tăng lên.

2.3.3 Set-Associative Cache



Hình 6: Cấu trúc Set-Associative Cache

Để hạn chế hiện tượng bottleneck xảy ra ở cấu trúc Direct Mapping Cache, cấu trúc Set-Associative Cache sẽ chia Cache ra thành nhiều "**set**". Nhờ đó, các block dữ liệu cùng được map vào cùng một cache line có thể được lưu trữ trong các **set**" khác nhau.

2.3.4 Replacement Policy

Để đơn giản hóa việc thiết kế, replacement policy chính được sử dụng trong thiết kế Cache được chọn là RANDOM. Giải thuật của RANDOM replacement policy không yêu cầu phải lưu trữ bất kì thông tin nào về lịch sử truy cập bộ nhớ.

Một lựa chọn khác được hỗ trợ là giải thuật Approximately Least-Recently-Used (ALRU), ALRU sẽ tìm xấp xỉ way ít được sử dụng nhất để thay thế do đó nó cần lưu trữ thông tin về lịch sử truy cập bộ nhớ. Cần chý ý rằng RVS192 Cache chỉ hỗ trợ cho 4-way ALRU.

2.3.5 Read Policy

Read hit: Đọc dữ liệu trực tiếp từ block quản lý nó ở level cache hiện hành.

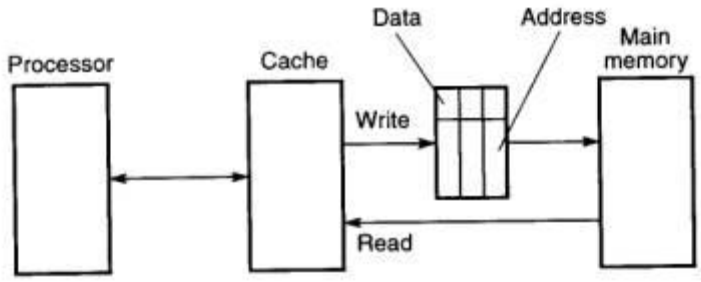
Read miss: Yêu cầu dừng Pipeline đồng thời yêu cầu dữ liệu từ level thấp hơn. Pipeline chỉ được phép hoặc động trở lại khi dữ liệu yêu cầu hợp lệ ở level cao nhất. Cache sẽ chọn một set trong line dữ liệu hiện hành để thay thế (replace) block dữ liệu cũ bằng block dữ liệu mới từ level thấp hơn. Nếu block bị thay thế dirty (level cao chứa dữ liệu mới, level thấp chứa dữ liệu cũ hơn), block dirty sẽ được đưa xuống level thấp hơn trước khi dữ liệu mới được đưa lên level hiện hành.

2.3.6 Write Policy

Write Policy được sử dụng cho RVS192 Cache là Write Back tích hợp Write Buffer:

Write hit: Ghi dữ liệu trực tiếp vào block quản lý nó ở level cache hiện hành. Block chứa dữ liệu được ghi vào này sẽ bị dirty (level cao chứa dữ liệu mới, level thấp chứa dữ liệu cũ hơn). Dirty block sẽ được đưa xuống level thấp hơn khi và chỉ khi nó bị thay thế (replace).

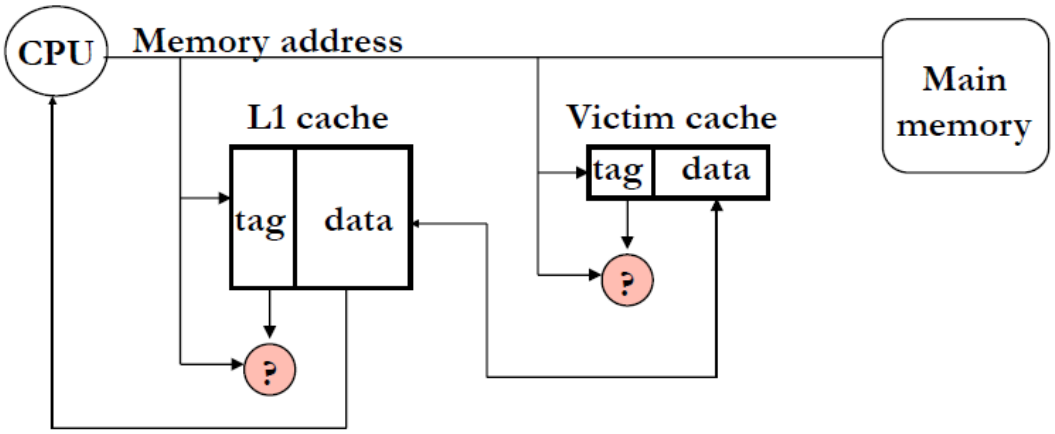
Write miss: Ghi dữ liệu vào Write Buffer và tiếp tục thực hiện các câu lệnh tiếp theo. Write Buffer liên tục yêu cầu gửi dữ liệu xuống level thấp hơn nếu nó đang chứa dữ liệu. Khi Write Buffer đầy, một yêu cầu ghi dữ liệu mới vào cache nếu bị miss sẽ yêu cầu dừng Pipeline chờ cho Write Buffer hết đầy.



Hình 7: Hình minh họa Write Buffer trong cấu trúc 1 level Cache

2.3.7 Victim Cache

Temporal locality xác định rằng khi một lệnh hay dữ liệu được truy cập, có nhiều khả năng lệnh hay dữ liệu đó sẽ được sử dụng lại trong tương lai. RVS192 Cache tận dụng tính chất này để tăng nhẹ hit-rate bằng cách thêm vào Victim Cache chứa dữ liệu vừa bị thay thế từ level 1 Cache. Do tính chất khác nhau của instruction và data nên Victim Cache của L1 Inst Cache và L1 Data Cache sẽ có một chút khác nhau. Cụ thể Victim Cache của Data sẽ được thêm bit dirty để đảm bảo dữ liệu được đọc ra là chính xác.



Hình 8: Minh họa cho Victim Cache trong cấu trúc 1 level Cache

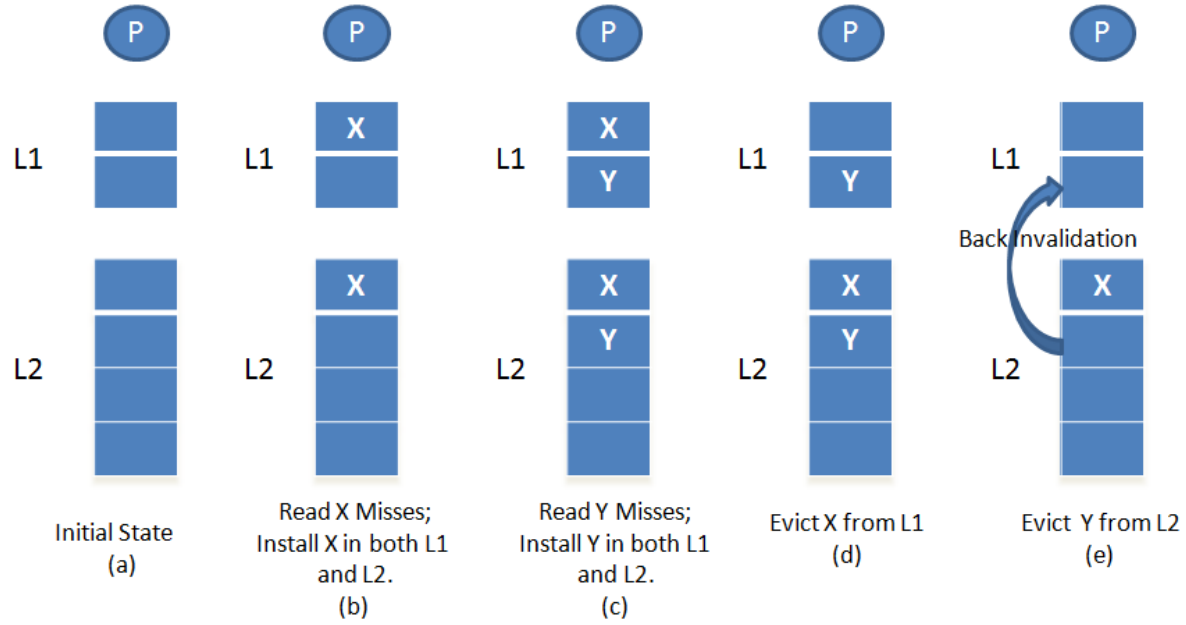
2.3.8 Cache Inclusion Policy

Đối với hệ thống Cache có nhiều hơn một Level Cache, một đặc tính cần phải quan tâm đó là Cache inclusion policy (Inclusive Policy, Exclusive Policy, NINE Policy). Cache inclusion policy được dử dụng trong RVS192 Cache là Inclusive Policy:

Tính chất này xác định rằng tất cả các block đang tồn tại trong level cao hơn phải nằm trong level thấp hơn nó.

Tại sao lại có yêu cầu này?

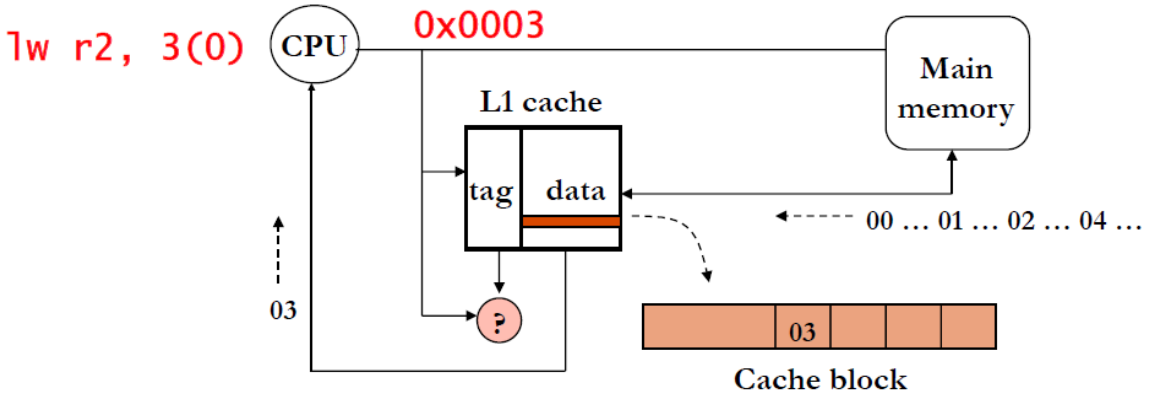
Giả sử chúng ta hiện thực một hệ thống gồm 2 level Cache, với các đặc tính được đề cập ở các mục trên. Khi một Block bị dirty từ Level 1 bị replace, block này sẽ được đưa xuống Level 2 Cache. Nếu hệ thống Cache không hỗ trợ Inclusive Policy, L2 Cache phải biết được rằng liệu nó có chứa block này không, nếu không thì phải ghi block này xuống tầng Memory thấp hơn nó. Với Inclusive Policy, block này sẽ này lập tức được ghi vào L2 cache, do nó chắc chắn sẽ có trong L2 Cache. Inclusive Policy yêu cầu L2 cache phải kiểm tra xem Block bị thay thế trong L2 Cache có nằm trong Level 1 Cache không, nếu có thì L1 Cache phải kiểm tra block đó có bị dirty không, dirty thì ghi xuống Level 2 Cache, sau đó loại bỏ block này (evict), không dirty thì loại bỏ trực tiếp. Vì truy cập xuống level càng thấp thì miss penalty sẽ càng cao do đó Inclusive Policy sẽ được lựu chọn.



Hình 9: Inclusive Policy

2.3.9 Critical Word First

Block dữ liệu chứa càng nhiều word thì miss-penalty sẽ càng tăng lên do mỗi khi có miss xảy ra, Cache phải lấy toàn bộ các word trong block dữ liệu bị miss. Để giảm miss-penalty, kỹ thuật Critical Word First (CWF) sẽ được sử dụng. CWF sẽ ưu tiên đưa word bị miss lên trước, các word còn lại sẽ được cập sau.



Hình 10: Minh họa cho kỹ thuật Critical Word First

2.3.10 Hit Under Miss

Kỹ thuật Hit Under Miss hỗ trợ cpu tiếp tục hoạt động nếu như ngay sau khi dữ liệu bị miss được đưa lên level cao nhất (các word khác trong block hiện hành vẫn đang chờ cập nhập), các truy cập tiếp theo vào Cache đều là cache hit thì Pipeline sẽ được phép hoạt động bình thường.

Kết hợp với kỹ thuật CWF được đề cập ở mục trên, Pipeline sẽ ngay lập túc được hoạt động trở lại khi có yêu cập nhập được gửi tới Level 1 Cache.

3. THIẾT KẾ VÀ THỰC HIỆN

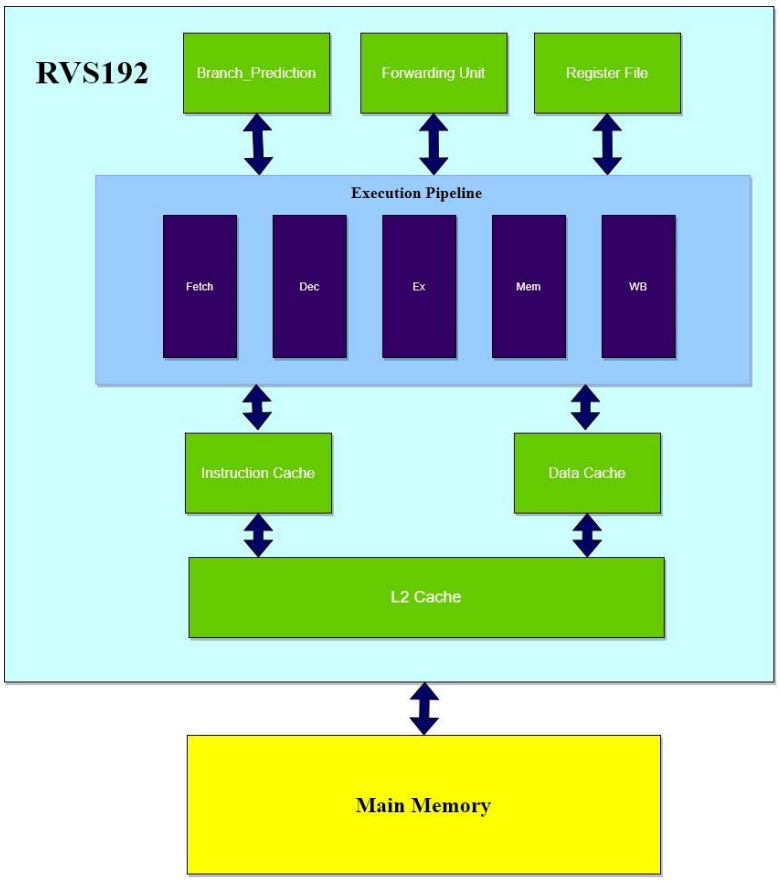
3.1 Đặc điểm hỗ trợ

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



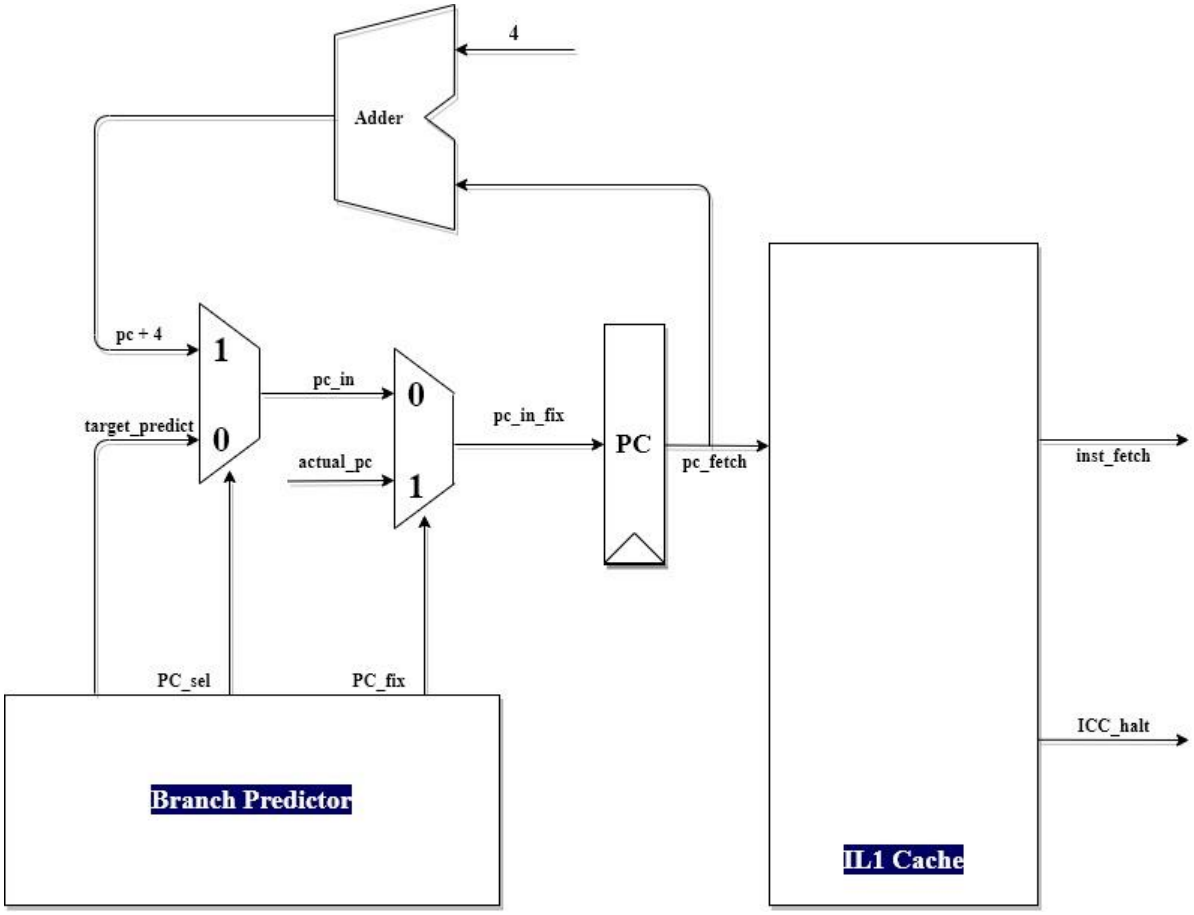
Hình 11: Cấu trúc tổng quát của lõi RVS192

3.2 Các tầng Pipeline

Chú ý: Các tín hiệu tại mỗi tầng pipeline đều có thêm các tiền tố quy định của tầng quản lý nó, một số tín hiệu đặc biệt được truy cập trực tiếp sẽ không có tiền tố thêm vào này. Tiền tố thêm vào theo nguyên tắc: tín hiệu từ thanh ghi pipeline đi ra thêm o_, tín hiệu đi vào thanh ghi pipeline thêm i_, sau đó là tên tầng tương ứng. Các tín hiệu không giao tiếp với thanh ghi pipeline sẽ không có các tiền tố này (*). Trong hình mô tả, các tín hiệu thuộc đúng tầng đang được đề cập sẽ không thể hiện tiền tố (có trong RTL code), các tín hiệu từ tầng khác sẽ có tiền tố tương ứng với tầng tạo ra nó.

Vd: i_pp_fetch_dec.pc; o_pp_dec_ex.pc

3.2.1 Fetch Stage



Hình 12: Fetch Stage

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

Tùy thuộc vào loại câu lệnh hiện hành và tín hiệu yêu cầu cập nhập từ tầng Execute, bộ Branch Predictor gửi các tín hiệu điều khiển việc chọn pc cho phù hợp.

- PC_sel (*): Nếu bộ Branch Predictor quyết định lệnh tiếp theo phải nhảy, tín hiệu này sẽ được kéo về 0 để chọn tín hiệu target_predict chứa địa chỉ nhảy tới sau lệnh hiện hành.
- PC_fix (*): Nếu có phát hiện lệnh nhảy được đoán sai từ tầng Execute, tín hiệu này sẽ được kéo lên 1 để chọn tín hiệu actual_pc chứa địa chỉ nhảy đến chính xác của lệnh bị đoán sai.

IL1 Cache (Level 1 Instruction Cache) sẽ kiểm tra xem câu lệnh tại địa chỉ pc_fetch có nằm trong nó hay không. Nếu không, IL1 Cache kéo tín hiệu ICC_halt lên mức cao yêu cầu dừng PC và các thanh ghi pipeline từ tầng Fetch sang Decode, đồng thời IL1 Cache cũng gửi tín hiệu yêu cầu cập nhập dữ liệu đến Level 2 Cache.

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

Tên tín hiệu	Mô tả
i_pp_fetch_dec.pc	pc
i_pp_fetch_dec.br_check	Tổ hợp các tín hiệu kiểm tra tính đúng đắn của lệnh nhảy
i_pp_fetch_dec.inst (aka: inst_fetch)	Câu lệnh cần xử lý

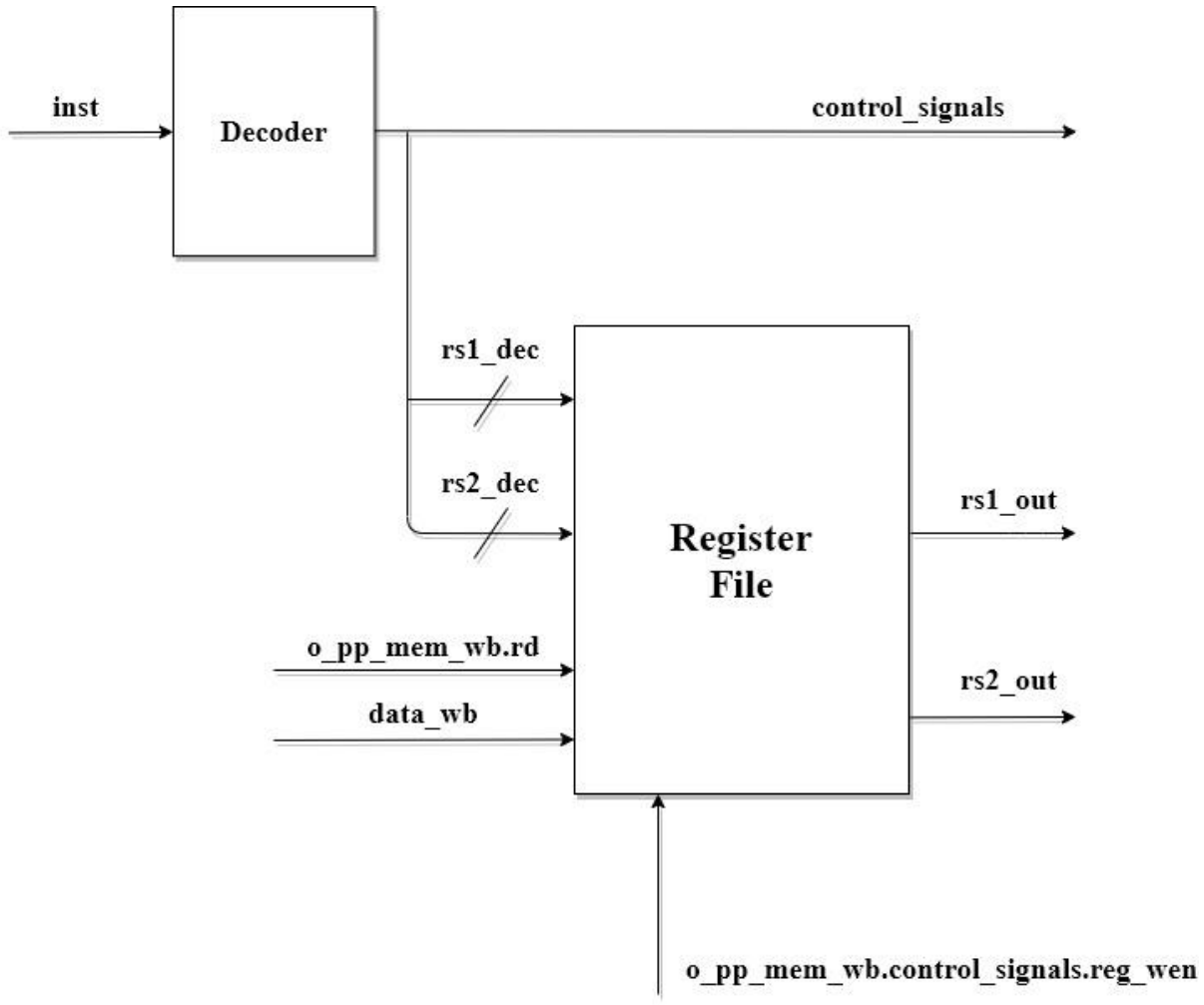
Bảng 1: Bảng mô tả tín hiệu từ tầng Fetch cần gửi đến các tầng sau

3.2.2 Decode Stage

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

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 Decoder sẽ giải mã các câu lệnh để đưa ra các tín hiệu điều khiển gửi tới các tầng sau. Bên cạnh đó, Decoder còn dùng để truy xuất giá trị immediate theo từng loại câu lệnh tương ứng.



Hình 13: Decode Stage

Lưu ý: data_wb là tín hiệu được đưa đến từ tầng Write Back

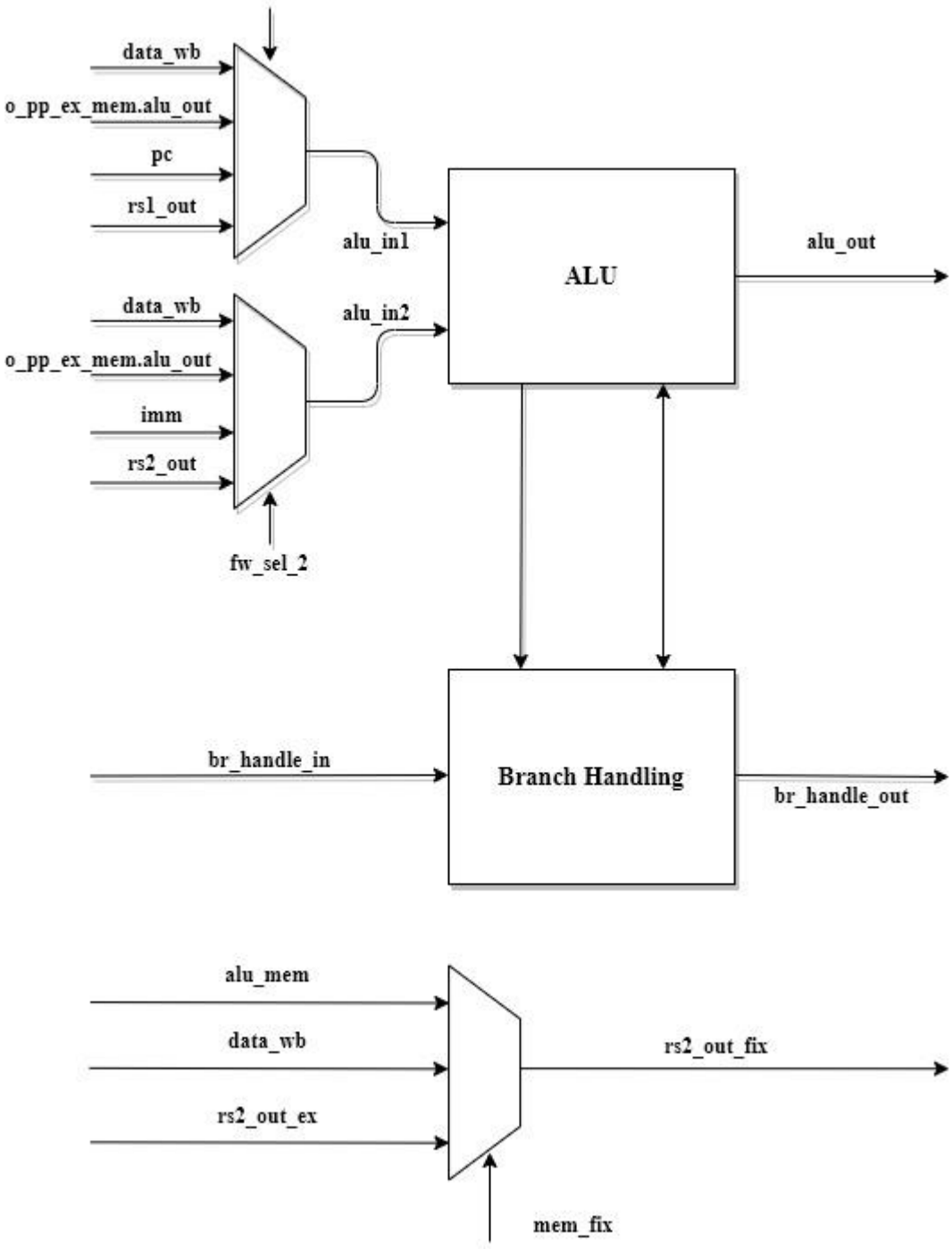
Danh sách các tín hiệu cần được truyền sang tầng tiếp theo:

Tên tín hiệu	Mô tả
i_pp_dec_ex.pc	pc
i_pp_dec_ex.br_check	Tổ hợp các tín hiệu kiểm tra tính đúng đắn của lệnh nhảy
i_pp_dec_ex.rs1	Địa chỉ thanh ghi rs1
i_pp_dec_ex.rs2	Địa chỉ thanh ghi rs2

i_pp_dec_ex.rd	Địa chỉ thanh ghi rd
i_pp_dec_ex.rs1_out	Giá trị thanh ghi rs1
i_pp_dec_ex.rs2_out	Giá trị thanh ghi rs2
i_pp_dec_ex.control_signals	Tổ hợp tín hiệu immediate và control được tạo ra từ bộ Decoder

Bảng 2: Bảng mô tả tín hiệu từ tầng Decode, cần gửi đến các tầng sau

3.2.3 Execute Stage



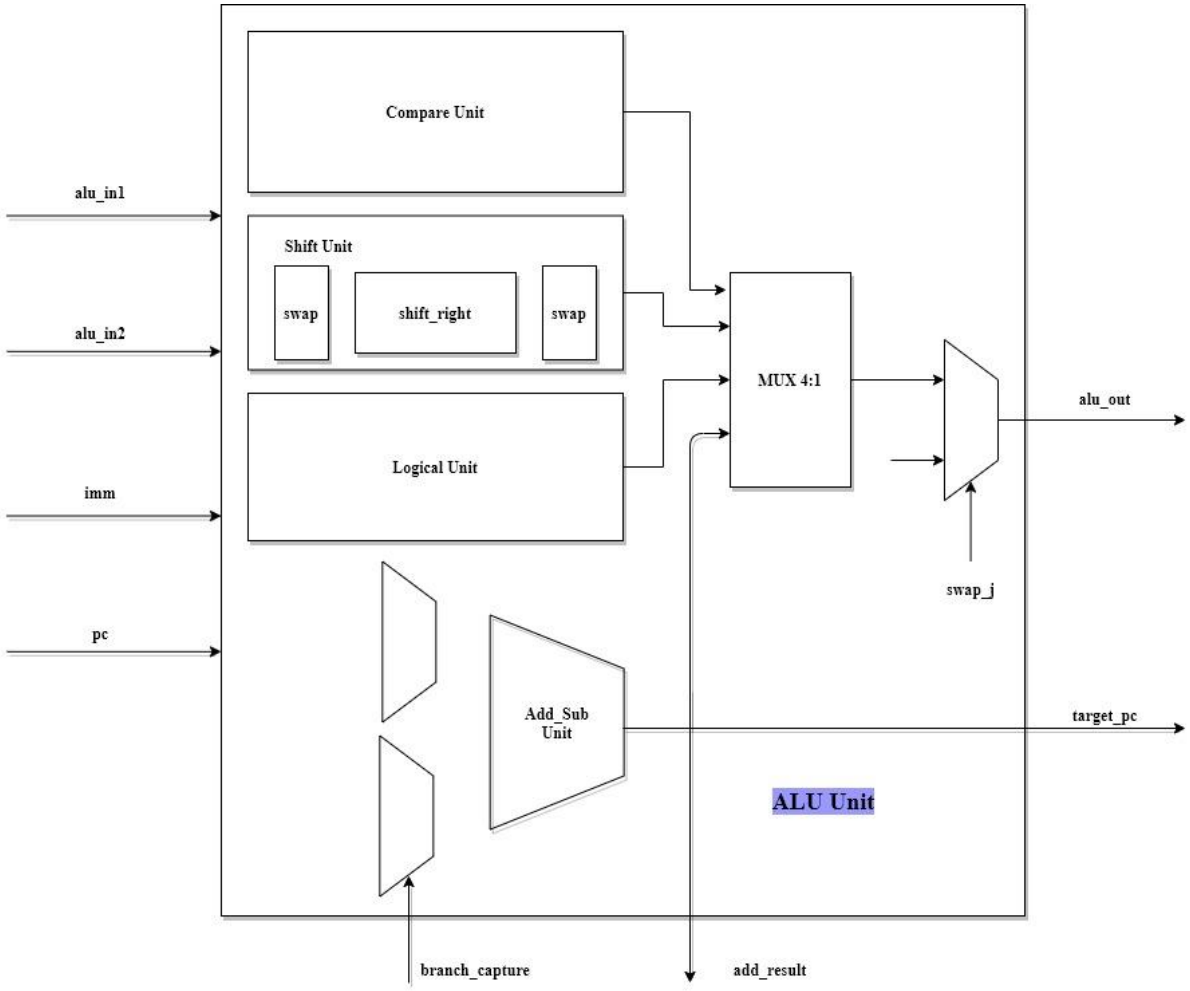
Hình 14: Execute Stage

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 15: ALU Unit

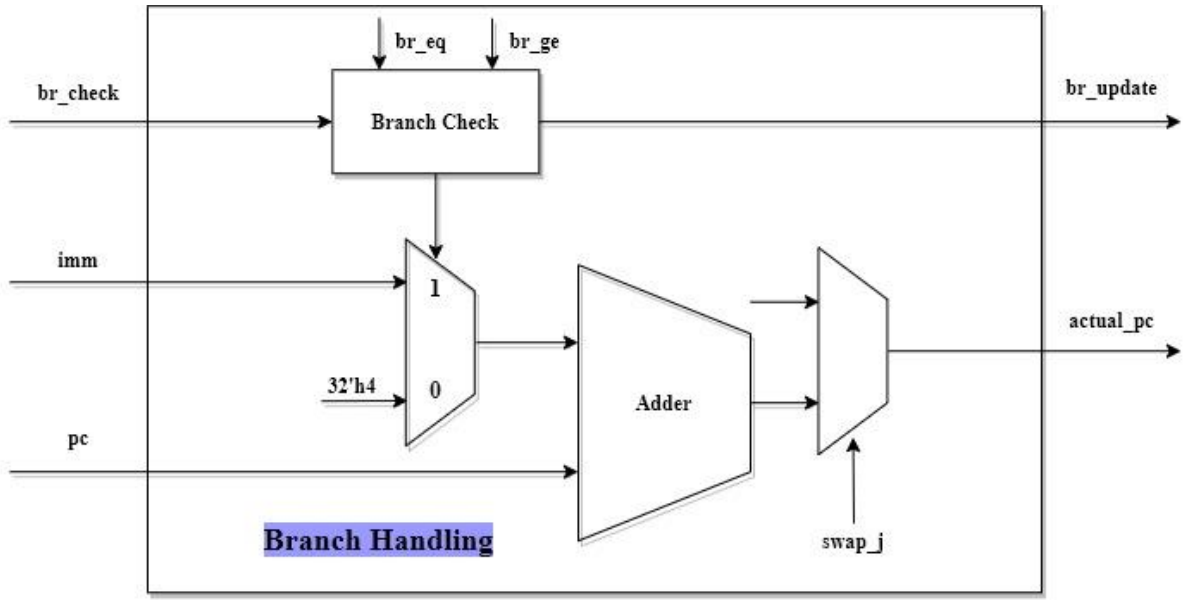
Dưới đây là bảng mô tả các câu lệnh sử dụng tài nguyên của bộ ALU

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	00	00
	sub	00	01
	lui	00	10
Shift Unit	sll/slli	01	00
	srl/srli	01	01
	sra/srai	01	10
Logical Unit	or/ori	10	00
	and/andi	10	01
	xor/xori	10	10

Compare Unit	slt/slti beq bne blt bge	11	00
	sltu/sltiu bltu bgeu	11	01

Bảng 3: 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 16: 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.

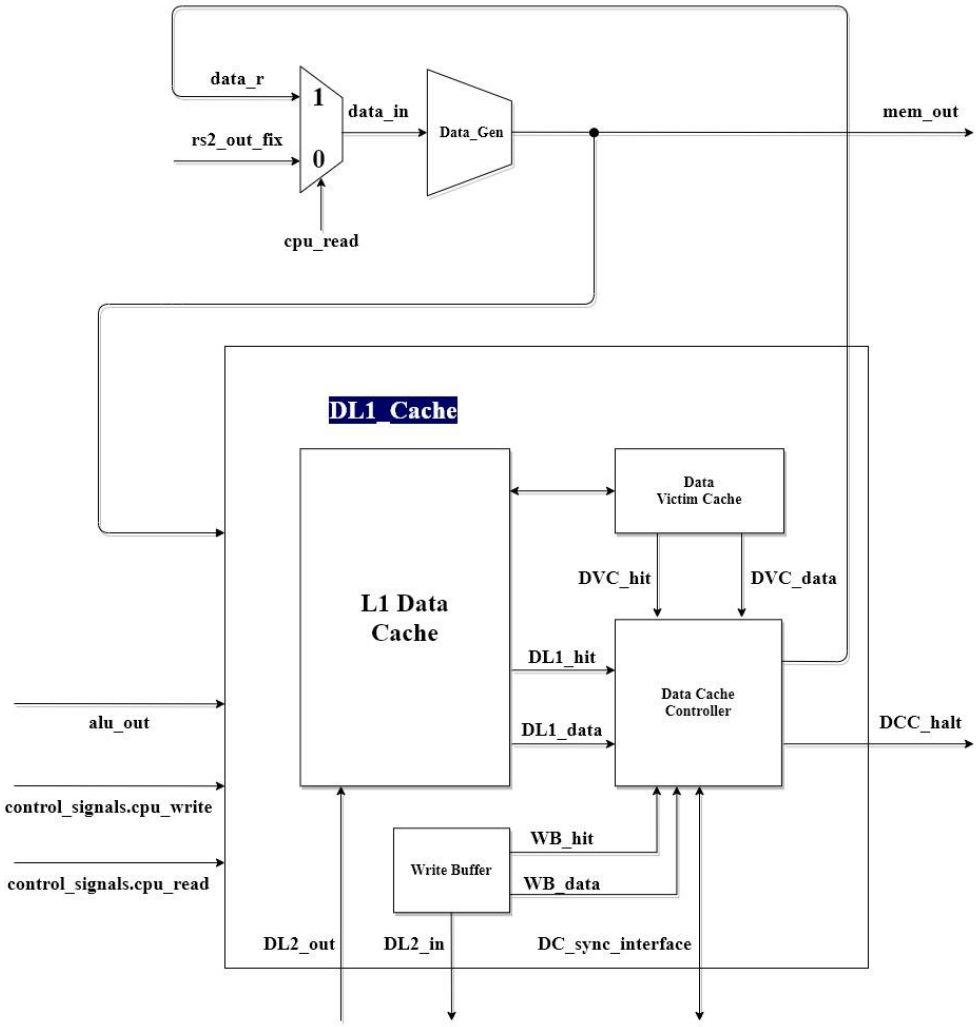
Danh sách các tín hiệu cần được truyền sang tầng tiếp theo:

Tên tín hiệu	Mô tả
i_pp_ex_mem.rd	Địa chỉ thanh ghi rd
i_pp_ex_mem.alu_out	Kết quả từ bộ ALU
i_pp_ex_mem.rs2_out_fix	Giá trị thanh ghi rs2 sau khi sửa các lỗi data hazard

i_pp_ex_mem.control_signals	Tổ hợp tín hiệu control cần gửi đến tầng Mem

Bảng 4: Bảng mô tả tín hiệu từ tầng Execute, cần gửi đến các tầng sau

3.2.4 Mem Stage



Hình 17: 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.

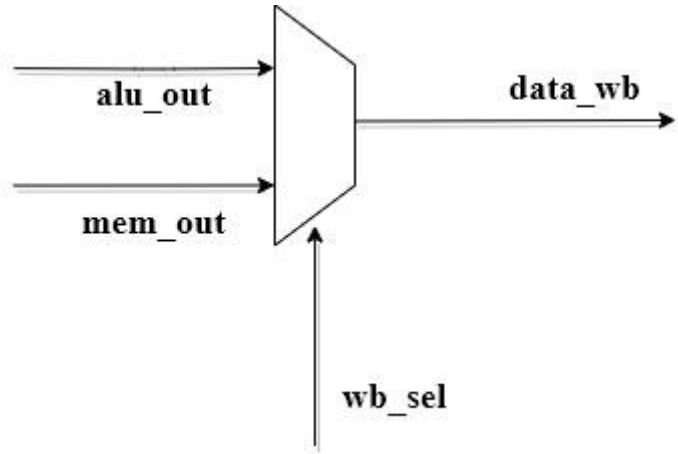
DL1 Cache (Level 1 Data Cache) sẽ kiểm tra xem câu lệnh tại địa chỉ pc_fetch có nằm trong nó hay không. Nếu không, DL1 Cache kéo tín hiệu DCC_halt lên mức cao yêu cầu dừng PC và các thanh ghi pipeline từ tầng Fetch sang Decode, đồng thời DL1 Cache cũng gửi tín hiệu yêu cầu cập nhập dữ liệu đến Level 2 Cache.

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

Tên tín hiệu	Mô tả
i_pp_mem_wb.rd	Địa chỉ thanh ghi rd
i_pp_mem_wb.alu_out	Kết quả từ bộ ALU
i_pp_mem_wb.mem_out	Dữ liệu đọc ra từ bộ nhớ
i_pp_mem_wb.control_signals	Tổ hợp tín hiệu control cần gửi đến tầng WB

Bảng 5: Bảng mô tả tín hiệu từ tầng Mem, cần gửi đến tầng WB

3.2.5 Write Back Stage

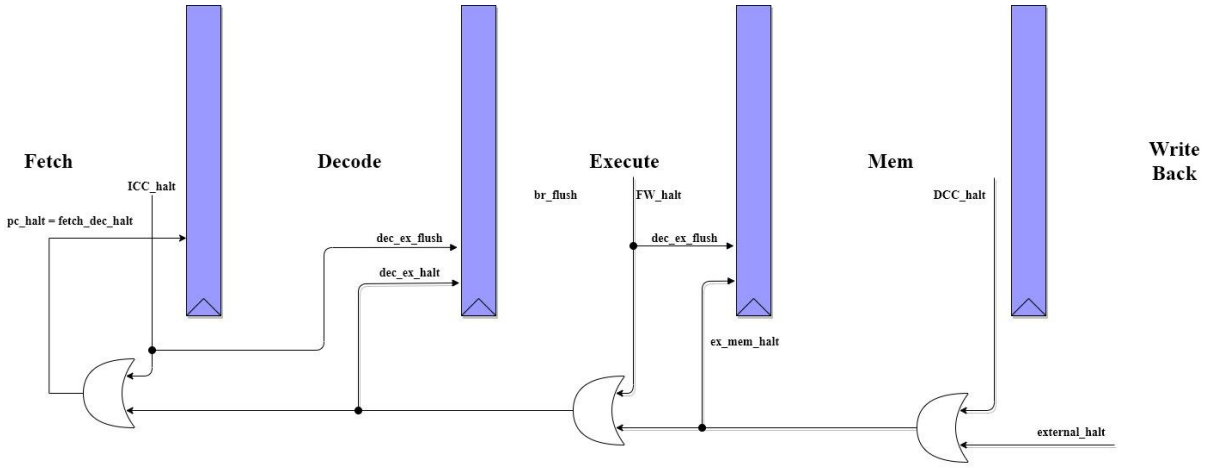


Hình 18: Write Back Stage

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.

3.3 Pipeline Register



Hình 19: Pipeline Register

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

- DCC_halt: tín hiệu từ Data Cache Controller.
- FW_halt: tín hiệu từ bộ Forwarding Unit.
- ICC_halt: tín hiệu từ Instruction Cache Controller.
- 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.

4. CẤU HÌNH RVS192

4.1 Cấu hình phần cứng hỗ trợ

Note: Chỉ hỗ trợ ALRU cho 4-way cache, với số lượng way khác 4 luôn sử dụng RANDOM

Macro	Mặc định	Mô tả
IL1 Cache		
INST_VICTIM_CACHE	Có	Tích hợp Victim Cache
ICACHE_ALRU (1)	Không	Sử dụng thuật toán thay thế ~LRU
ICACHE_RANDOM (1)	Có	Sử dụng thuật toán thay thế RANDOM

DL1 Cache		
DATA_VICTIM_CACHE	Có	Tích hợp Victim Cache
DCACHE_ALRU [(2)]	Không	Sử dụng thuật toán thay thế ~LRU
DCACHE_RANDOM [(2)]	Có	Sử dụng thuật toán thay thế RANDOM
Branch Predictor		
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 6: 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

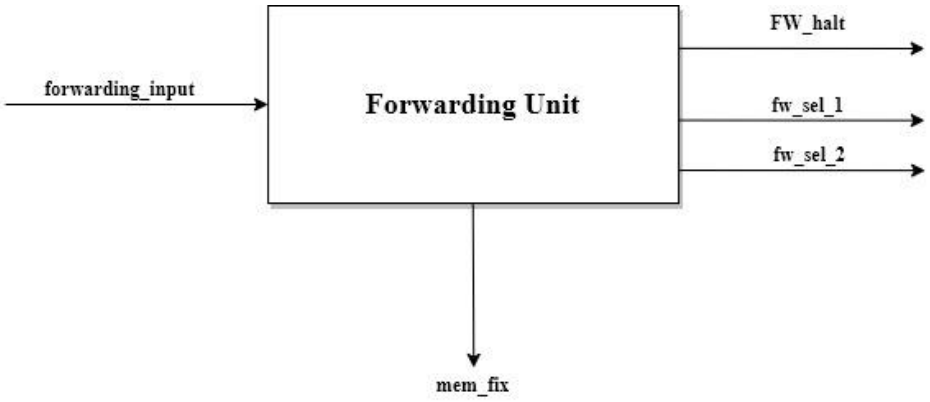
RVS192 được thiết kế theo cấu trúc RISC-V cpu 32-bit do đó parameter INST_LENGTH được cố định là 32 và không được phép thay đổi bởi người dùng.

Parameter	Mặc định	Mô tả
CACHE		
CACHE_BLOCK_SIZE	64	Số byte được lưu trữ trong một Line
IL1 Cache		
ICACHE_LINE	128	Số lượng Line trong mỗi Way
ICACHE_WAY	2	Instruction Cache associativity

DL1 Cache		
DCACHE_LINE	32	Số lượng Line trong mỗi Way
DCACHE_WAY	4	Data Cache associativity
DCACHE_WB_DEPTH	10	Dung lượng của Data Cache Write Buffer
L2 Cache		
L2_CACHE_LINE	64	Số lượng Line trong mỗi Way
L2_CACHE_WAY	8	L2 Cache associativity
L2_CACHE_WB_DEPTH	16	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_INDEX	4096	Số bit được sử dụng làm index cho GPT của bộ Gshare Branch Predictor
LOCAL_LPT_INDEX	1024	Số bit được sử dụng làm index cho LPT của bộ Local Branch Predictor
LOCAL_LHT_INDEX	1024	Số bit được sử dụng làm index cho LHT của bộ Local Branch Predictor
BTB_INDEX	8	Số bit được sử dụng làm index cho BTB
BTB_TAG_LENGTH	30 - BTB_INDEX	Số bit được lưu làm Tag trong BTB

Bảng 7: Bảng parameter cấu hình phần cứng

5. FORWARDING UNIT

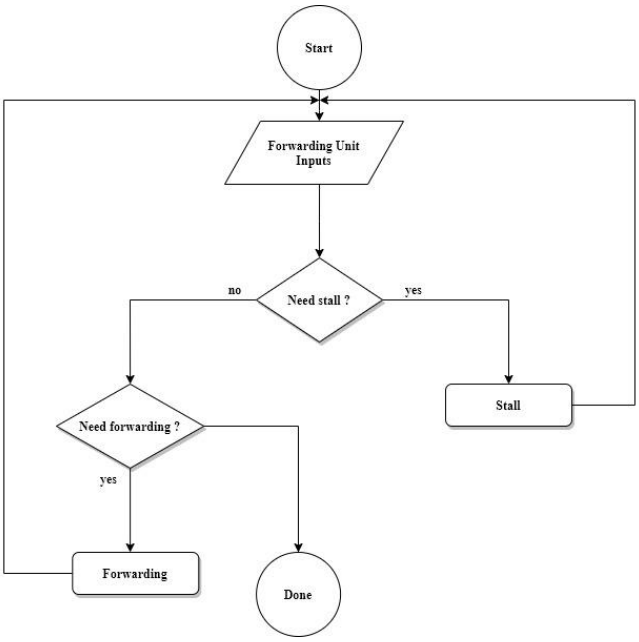


Hình 20: Forwarding Unit

Bộ Forwarding Unit làm nhiệm vụ sửa lỗi Data Hazard:

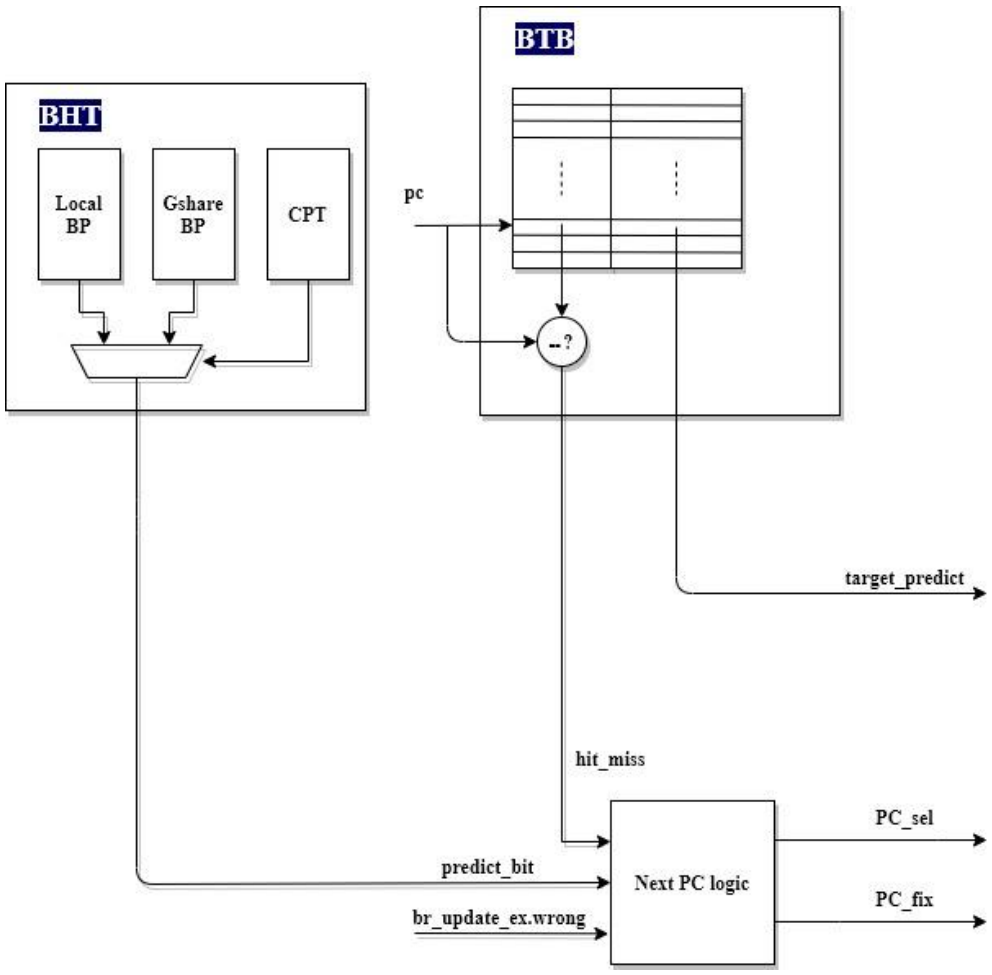
- Các lệnh chỉ sử dụng thanh ghi (R-Format): lỗi data hazard được giải quyết bằng cách forwarding kết quả đã hoàn thành ở các tầng sau để cung cấp cho tầng phía trước. Pipeline vẫn hoạt động xuyên suốt mà không cần phải dừng lại.
- Các lệnh truy cập data memory (một phần của I-Format và S-Format): các 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 21: Forwarding Unit Algorithm

6. BRANCH PREDICTION UNIT



Hình 22: 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.

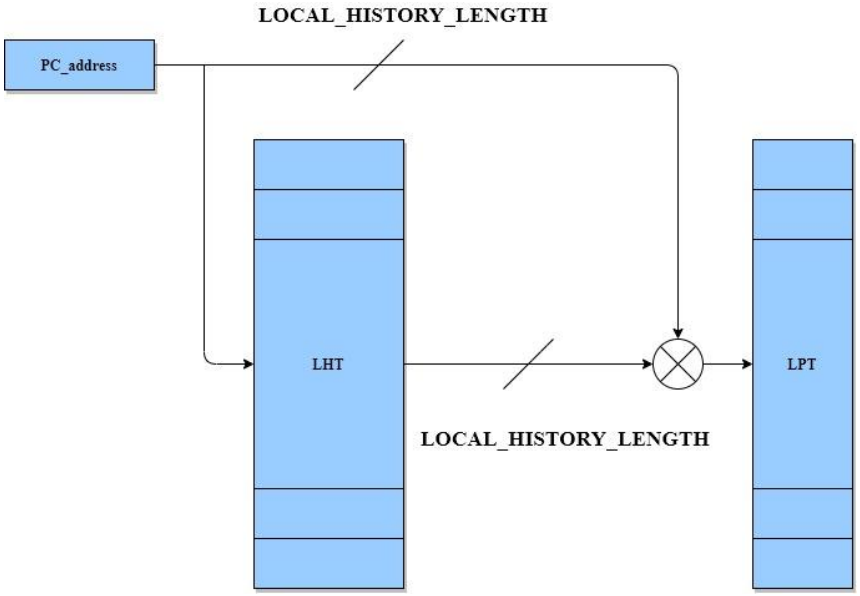
Bộ BPU gồm 3 khối chính được thể hiện trên Hình 6.1:

- 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 được gửi đến từ tầng Execute, 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 Predictor



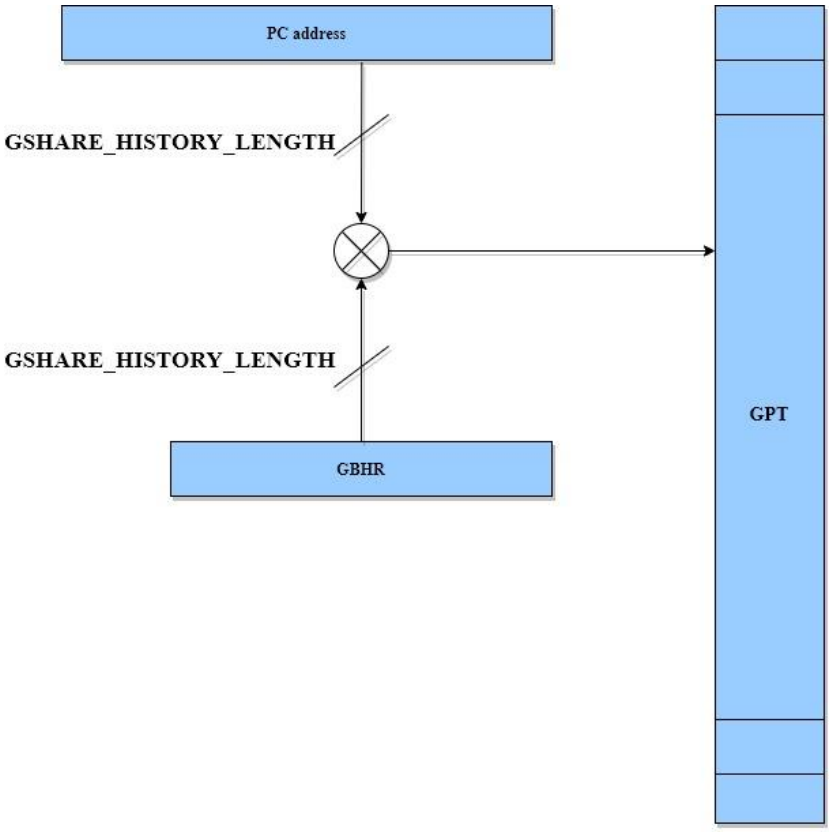
Hình 23: BHT với cấu hình Local BP

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 Branch Handling ở tầng Execute. Bit cao của ô nhớ LPT tương ứng với lệnh nhảy sẽ được sử dụng làm predicted_bit.

6.2 Gshare Branch Predictor



Hình 24: 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:

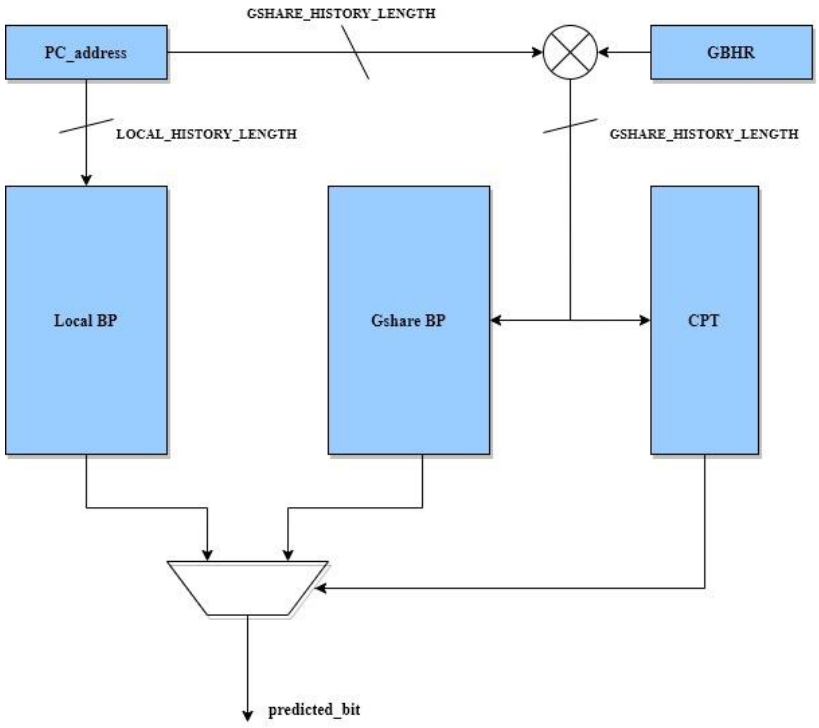
```
…       if(a)
            b = 1;
…
        if(b)
            …
```

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 Branch Handling ở tầng Execute. 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 Predictor



Hình 25: 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.

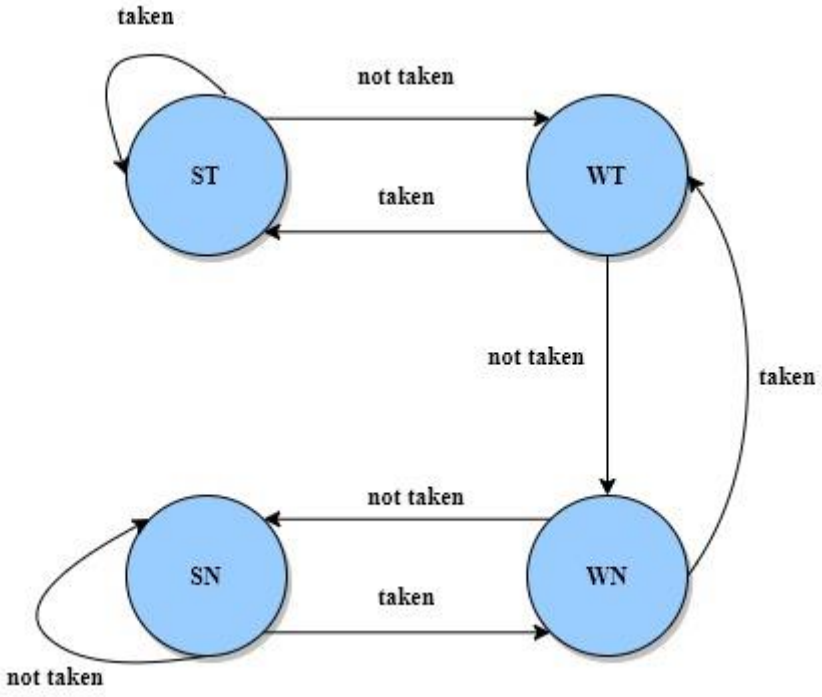
CPT (Choose Pattern Table) có cách hoạt động, lấy địa chỉ y hệt như GPT do đó thanh ghi GBHR của Gshare BP sẽ được đưa ra ngoài để dùng chung cho cả CPT và GPT. Mỗi ô nhớ GPT (Global Pattern Table) cũng chứa 2 bit predict được tạo ra từ bộ 2-level

predictor tích hợp trong khối Branch Handling ở tầng Execute. 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

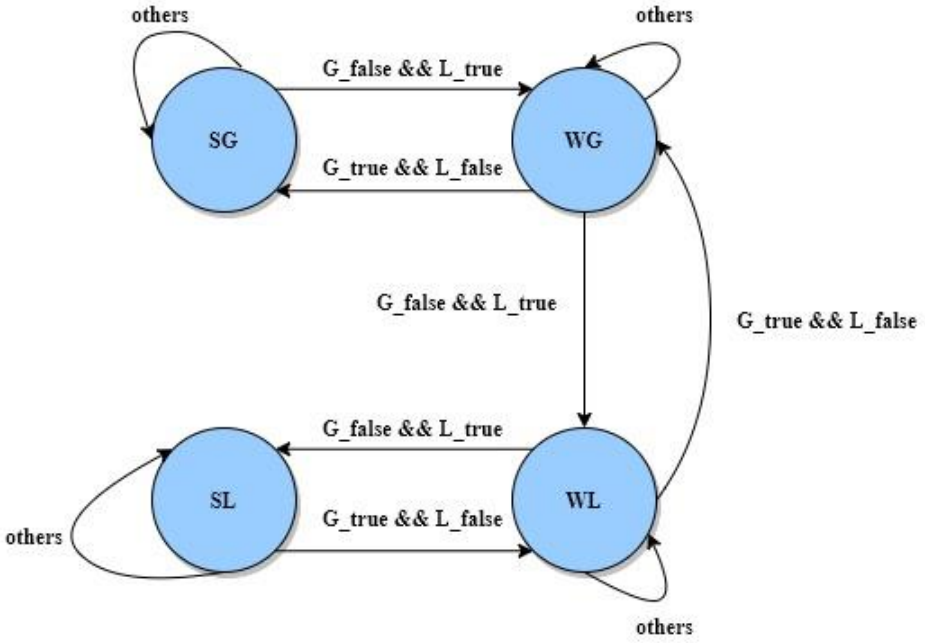
Bộ 2-Level Predictor thực chất là một FSM 4 trạng thái.

- 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.



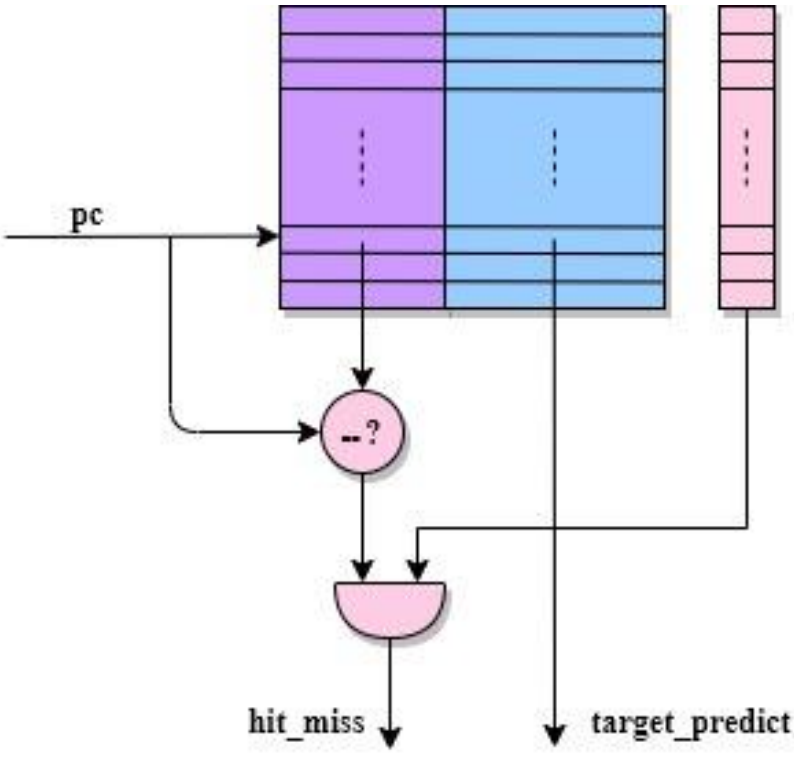
Hình 26: Sơ đồ chuyển trạng thái 2-bit Predictor của LPT và GPT

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

6.5 Branch Target Buffer



Hình 28: 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

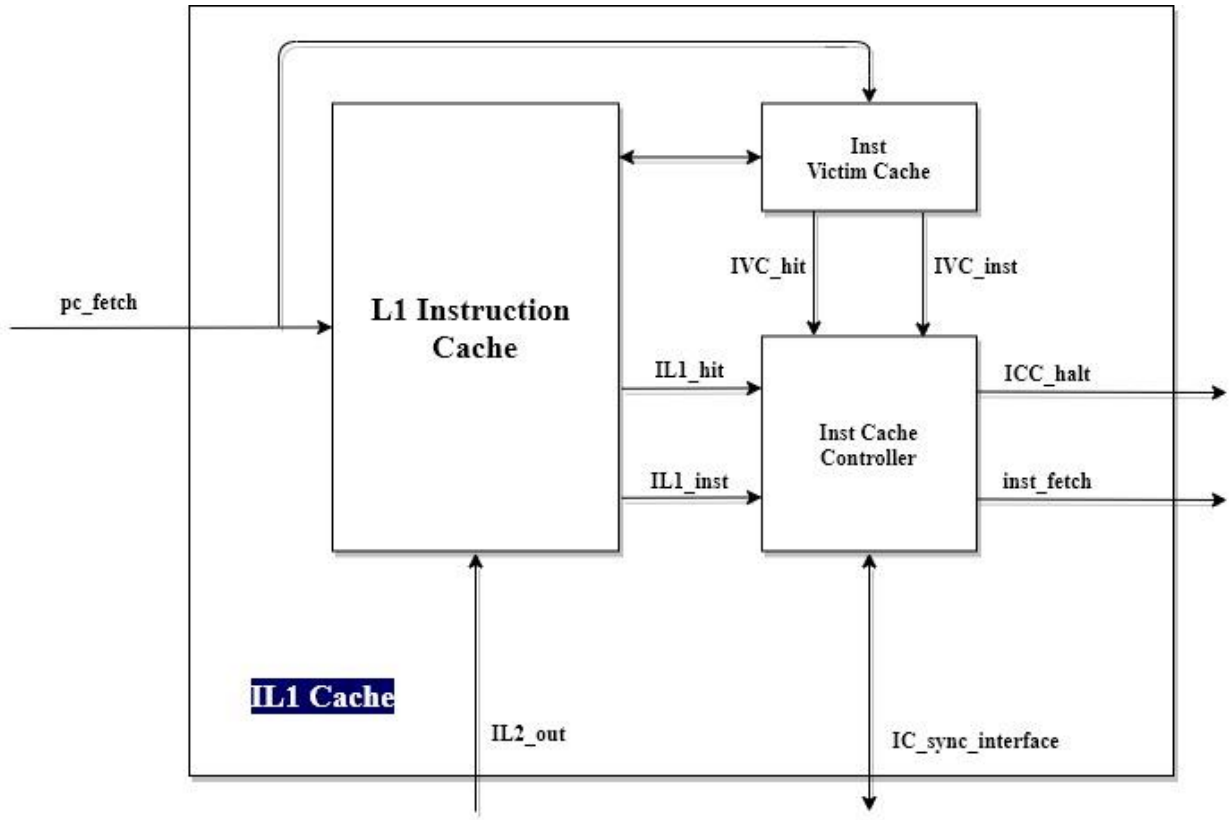
Phần này tập trung mô tả chức năng và cách thức hoạt động của Cache chứ không đi sâu vào chi tiết cấu trúc bên trong Cache. Tùy theo cách cài đặt, cấu trúc bên trong của Cache sẽ sinh ra các mạch khác nhau, tuy nhiên chức năng hoạt động vẫn tương tự. Dưới đây là bảng mô tả các thông số và cấu hình của 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: 16Kbyte, 64byte/block $D: 16Kbyte, 64byte/block	able
Associativity	$I: 2-way $D: 4-way	able
Updata Policy	ALRU, Random, Critical word first	unable
Data Cache Write Policy	Write Buffer, Write Back	unable
L2 CACHE		
Organization	Share Cache	unable
Size	64Kbyte, 64byte/block	able

Associativity	8-way	able
Updata Policy	Random, Critical word first	unable
Write Policy	Write Buffer, Write Back	unable

Bảng 8: Bảng mô tả cấu hình Cache của RVS192

7.1 IL1 Cache



Hình 29: IL1 Cache

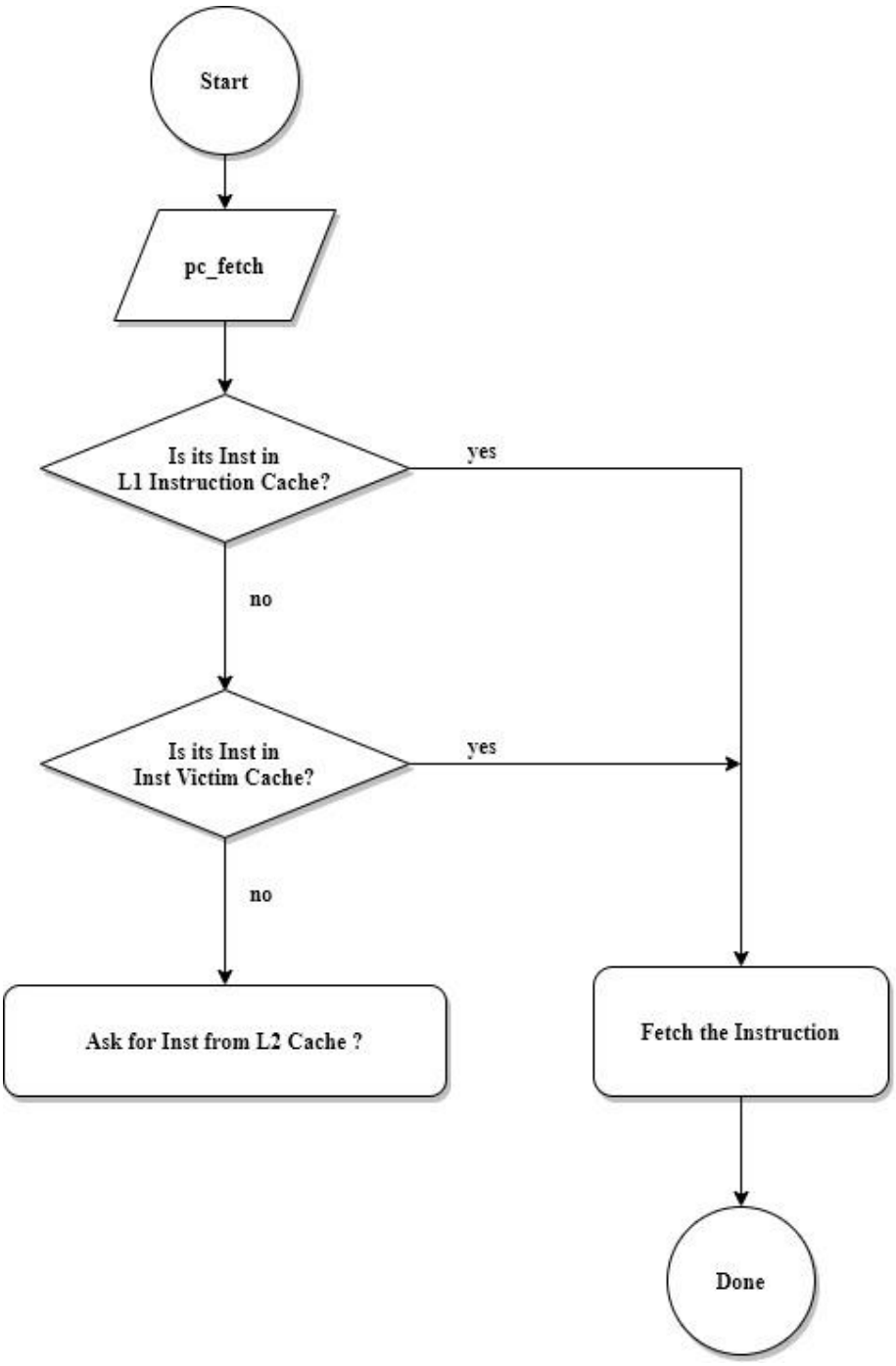
Như hình 7.1.1, ta có thể thấy IL1 Cache chứa 3 khối chính: L1 Instruction Cache, Inst Victim Cache và Inst Cache Controller. Trong đó:

- 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.

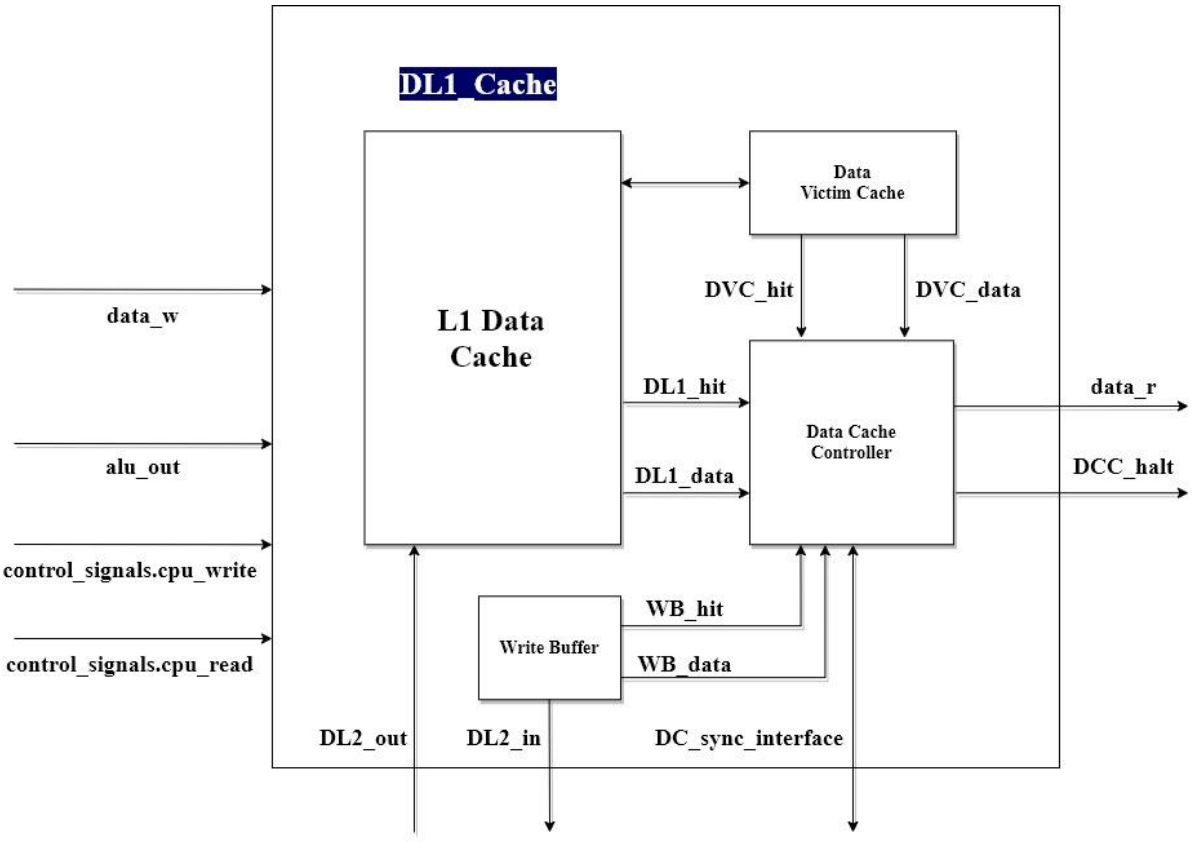
Giải thuật hoạt động của IL1 Cache được thể hiện trong hình bên dưới:



Hình 30: 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

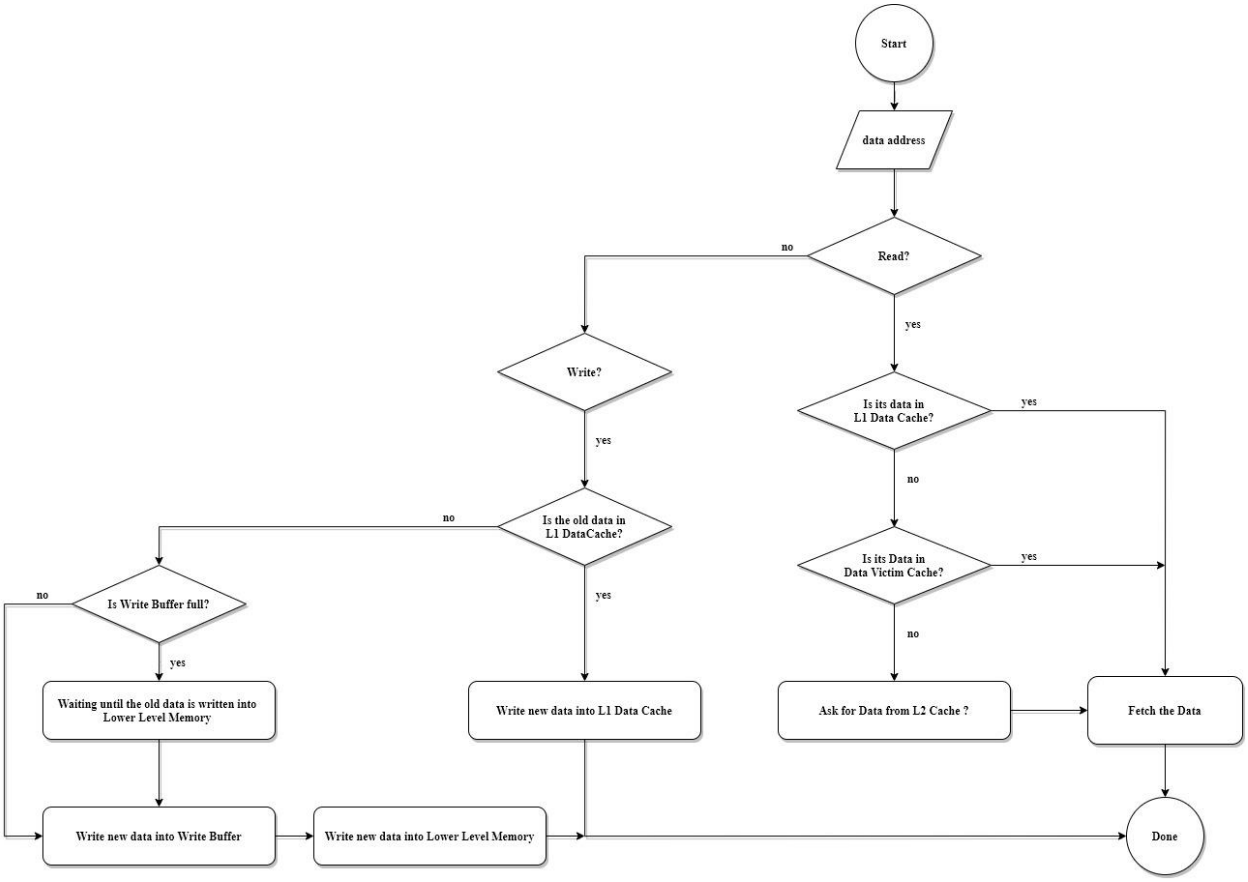


Hình 31: DL1 Cache

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:

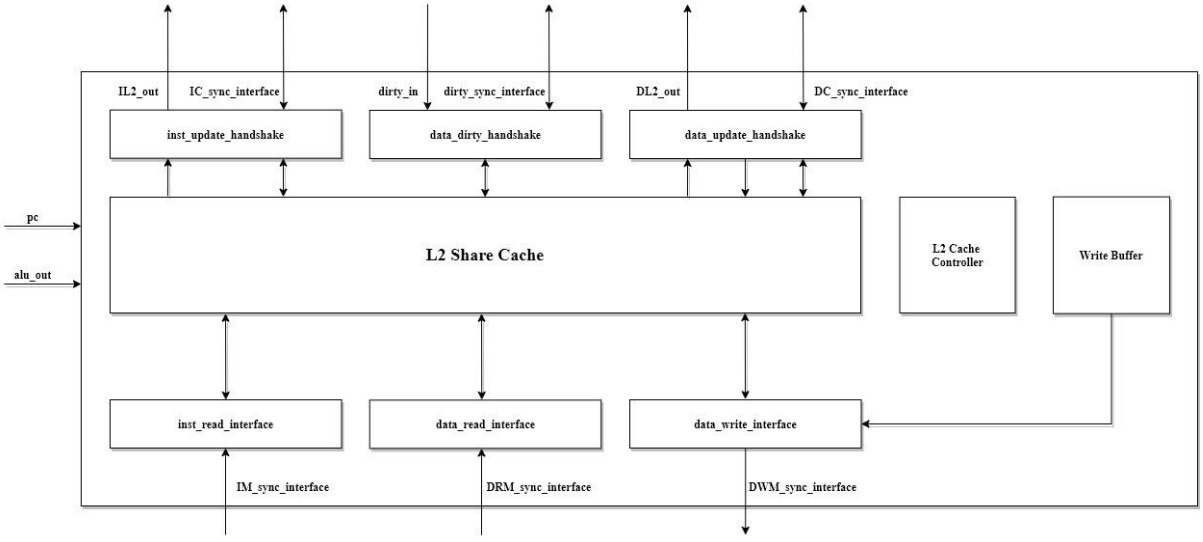
- L1 Data Cache: là bộ nhớ dữ liệu chính của DL1 Cache. Khi phát hiện ô nhớ tương ứng với địa chỉ hiện thời có trong bộ nhớ. Đối với lệnh đọc, L1 Data Cache sẽ tích cực tín hiệu DL1_hit và gửi câu lệnh tương ứng trên đường DL1_data đưa vào Data Cache Controller. Đối với lệnh ghi, L1 Data Cache cũng sẽ 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 DL1 Cache được thể hiện trong hình bên dưới:



Hình 32: Data Cache Controller Algorithm

7.3 L2 Cache



Hình 33: L2 Cache

L2 Cache chứa các bộ nhớ dữ liệu (L2 Share Cache), FIFO (Write Buffer), bộ điều khiển trung tâm (L2 Cache Controller) và các bộ giao tiếp đồng bộ giữa các miền clock.

Để hỗ trợ cấu hình Critical Word First, L2 Cache phải liên tục giám sát pc và alu_out. Nếu như block chứa dữ liệu tương ứng của pc và alu_out nằm trong L2 Share Cache và đang có yêu cầu update Level 1 Cache, dữ liệu tương ứng của pc và alu_out sẽ được ưu tiên đưa lên trước. Các word đã được gửi sẽ được bị dấu để hạn chế việc word đó được gửi lại, nhờ đó miss-penalty sẽ giảm.

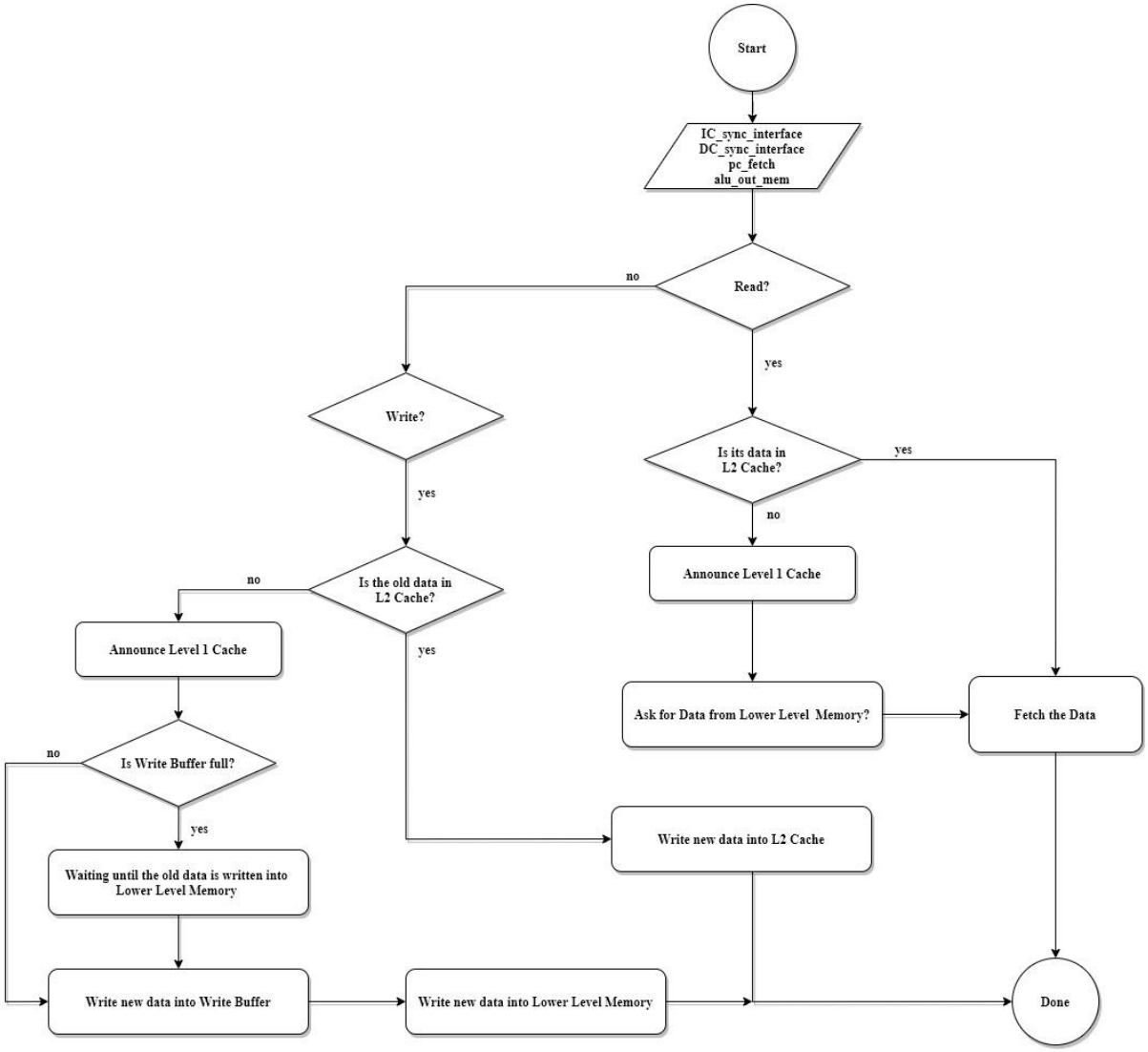
Để đơn giản thiết kế của bộ Cache, kỹ thuật *Hit Under Miss* chỉ được sử dụng ở Level 1 Cache.

Khi L2 Cache cần thay thế (replace) hay loại bỏ (evict) một block dữ liệu, trước hết nó phải gửi thông báo đến Level 1 Cache để đảm bảo đặc tính Inclusive Policy của hệ thống Cache.

Kỹ thuật đồng bộ được sử dụng:

- Bắt tay điều khiển (giao tiếp giữa Memory và L2 Cache).
- Điều khiển tín hiệu cho phép (giao tiếp giữa 2 Level Cache).

Giải thuật hoạt động của DL1 Cache sẽ được thể hiện trong hình bên dưới:



Hình 34: L2 Cache Algorithm

8. KẾT QUẢ THỰC HIỆN

8.1 Phần mềm sử dụng

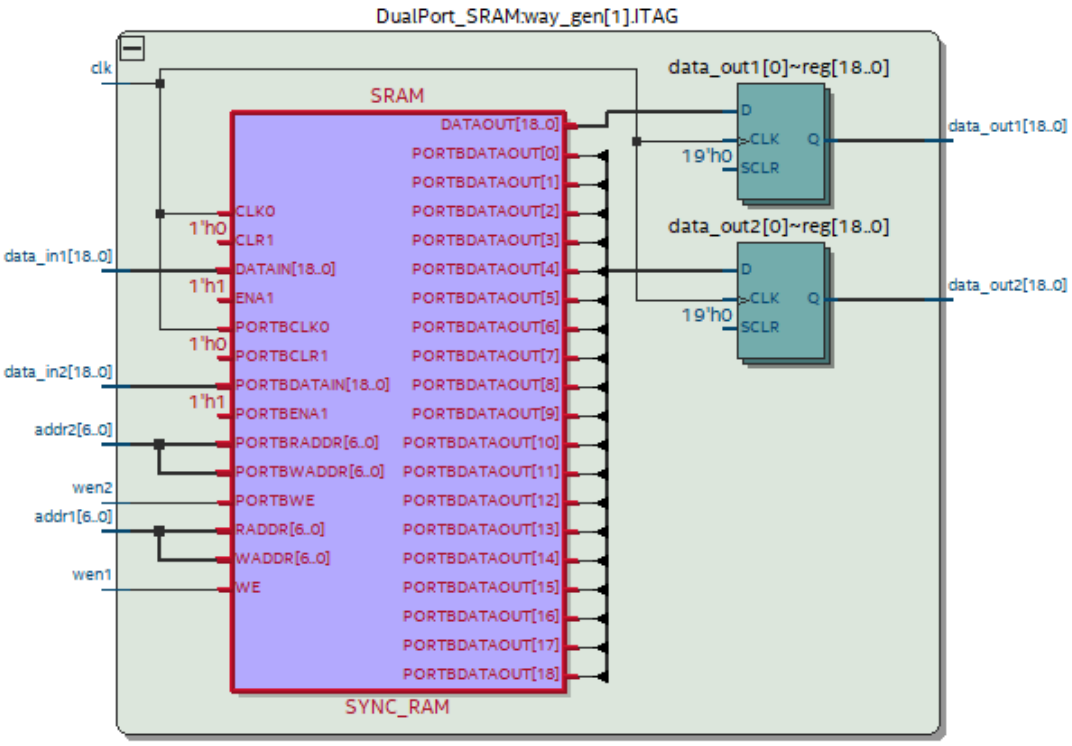
Phần mềm	Mục đích sử dụng
Microsoft Word 2016	Viết báo cáo, mô tả cấu trúc
drawio	Vẽ sơ đồ cấu trúc
Notepad++	Mô tả RTL code
QuestaSim	Mô phỏng RTL code
Quartus Prime Lite Edition 18.1	Khả tổng hợp RTL code
RARS 1.0	Tạo assembly code cho RVS192

Bảng 9: Phần mềm sử dụng

8.2 Quy trình xây dựng RVS192

8.2.1 Hiện thực Cache

Sau khi phân tích các yêu cầu thiết kế, cấu trúc Synchronous-Dual-Port Ram sẽ được dùng làm bộ nhớ dữ liệu cho toàn bộ cấu trúc của hệ thống Cache. Để đảm bảo việc ghi đọc dữ liệu là chính xác, một port của Dual Ram chỉ được dùng cho việc ghi dữ liệu mới và port còn lại sẽ được dùng để đọc dữ liệu.



Hình 35: Synchronous Dual Port Ram

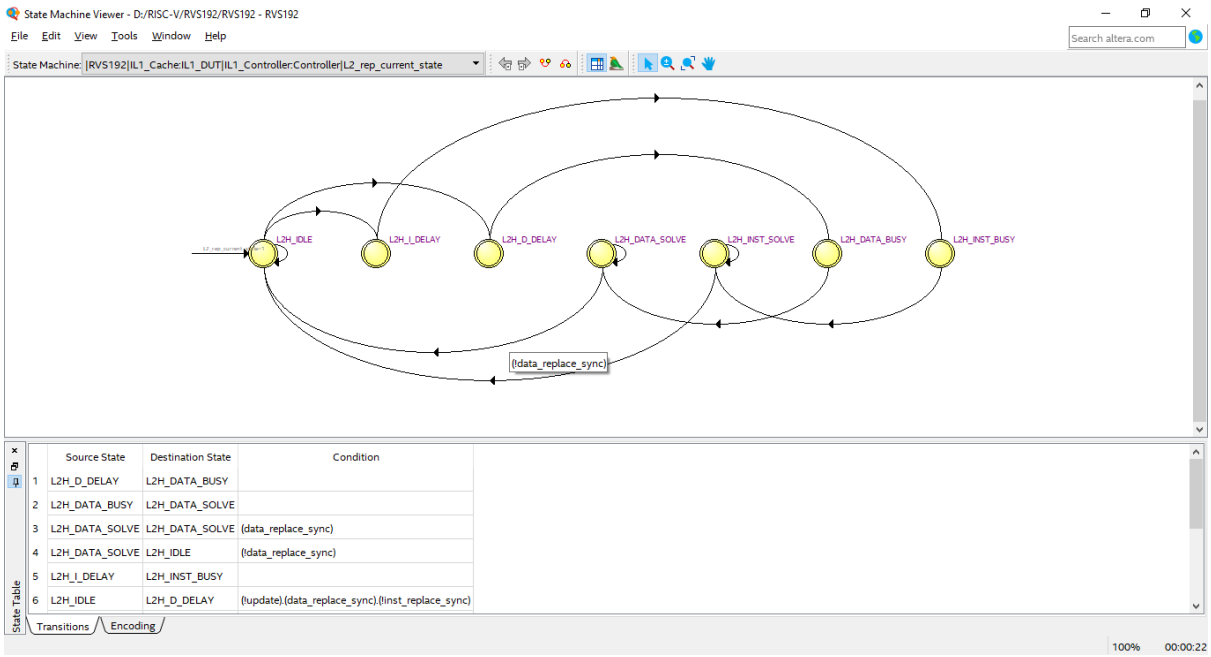
Tùy vào việc thiết lập các thông số ICACHE_WAY, DCACHE_WAY và L2_CACHE_WAY mà cấu trúc Cache sẽ là set-associative cache hay là direct-mapping cache.

8.2.1.1 IL1 Controller

IL1 Controller sẽ gồm 2 bộ FSM riêng biệt:

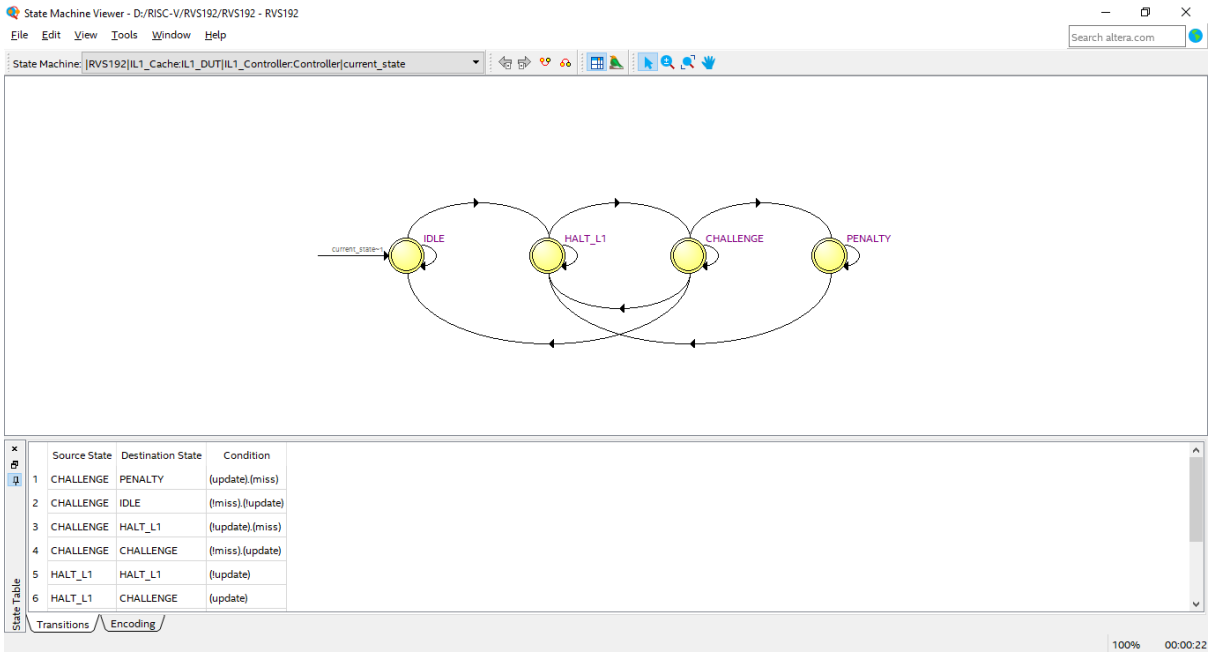
- IL1 inclusion FSM: đây là FSM dùng để đảm bảo tính chất inclusive của IL1 Cache. FSM này sẽ liên tục chờ tín hiệu kích hoạt từ L2 Cache. Nếu như có yêu cầu discard một line nhớ trong L2 Cache, tín hiệu yêu cầu *inst_replace_req* hoặc

data_replace_req được gửi tới IL1 Cache. Lúc này IL1 Cache sẽ kiểm tra line nhớ đó có thuộc sở hữu của mình không. Nếu có, IL1 inclusion FSM sẽ loại bỏ (discard) line nhớ, sau đó gửi *inst_replace_il1_ack* để báo cho L2 Cache. Nếu không, nó chỉ gửi *inst_replace_il1_ack*. Trường hợp có 2 yêu cầu discard cùng lúc do L2 Cache yêu cầu thay thế line nhớ (để lưu cả inst và data mới), FSM này sẽ xử lý tuần tự inst rồi đến data.



Hình 36: IL1 inclusion FSM

- IL1 FSM: đây là FSM sẽ quản lý việc cập nhập dữ liệu mới cho IL1 Cache. Khi có miss xảy ra, IL1 FSM sẽ vào trạng thái HALT_L1, tích cực tín hiệu ICC_halt cho đến khi nhận được dữ liệu mong muốn từ L2 Cache. Sau word đầu tiên nhận được kể từ khi có miss xảy ra, IL1 FSM vào trạng thái CHALLENGE, tắt yêu cầu ICC_halt. Nếu như tất cả các lệnh tiếp theo đều được hit, thì IL1 FSM sẽ chờ đến khi cập nhập dữ liệu xong thì về IDLE. Nếu như trong thời gian chờ cập nhập đó có miss xảy ra, IL1 FSM về trạng thái PENALTY, yêu cầu dừng pipeline đến khi cập nhập dữ liệu xong và nhảy tới HALT_L1.

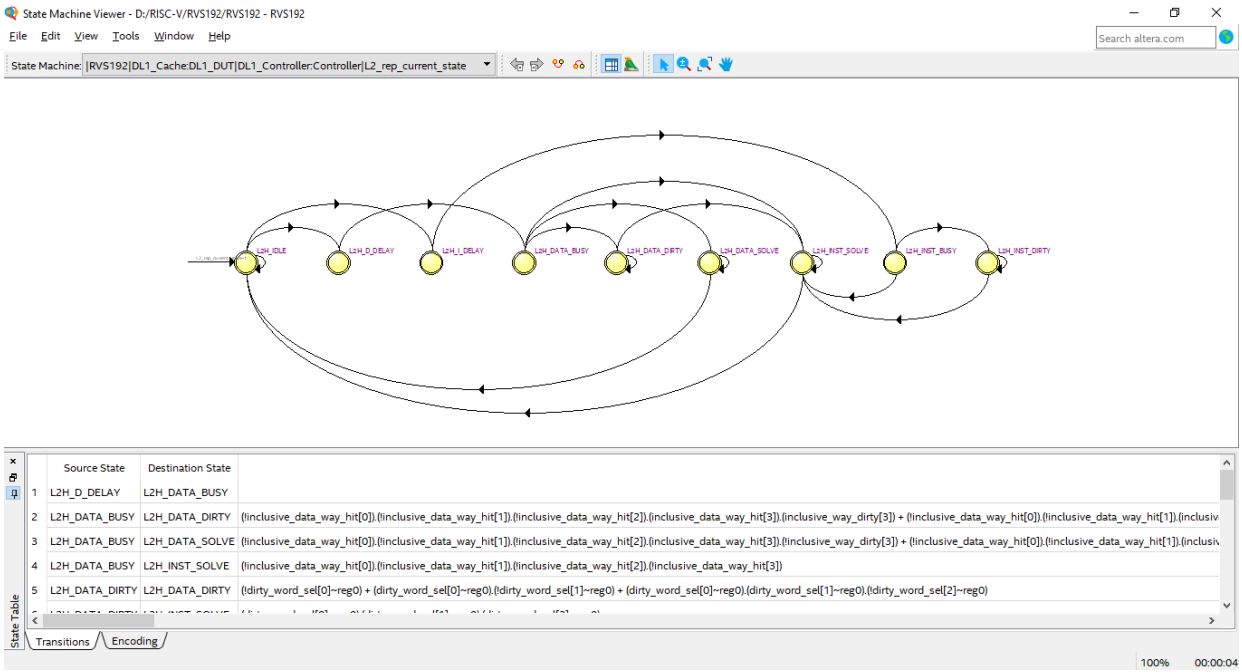


Hình 37: IL1 FSM

8.2.1.2 DL1 Controller

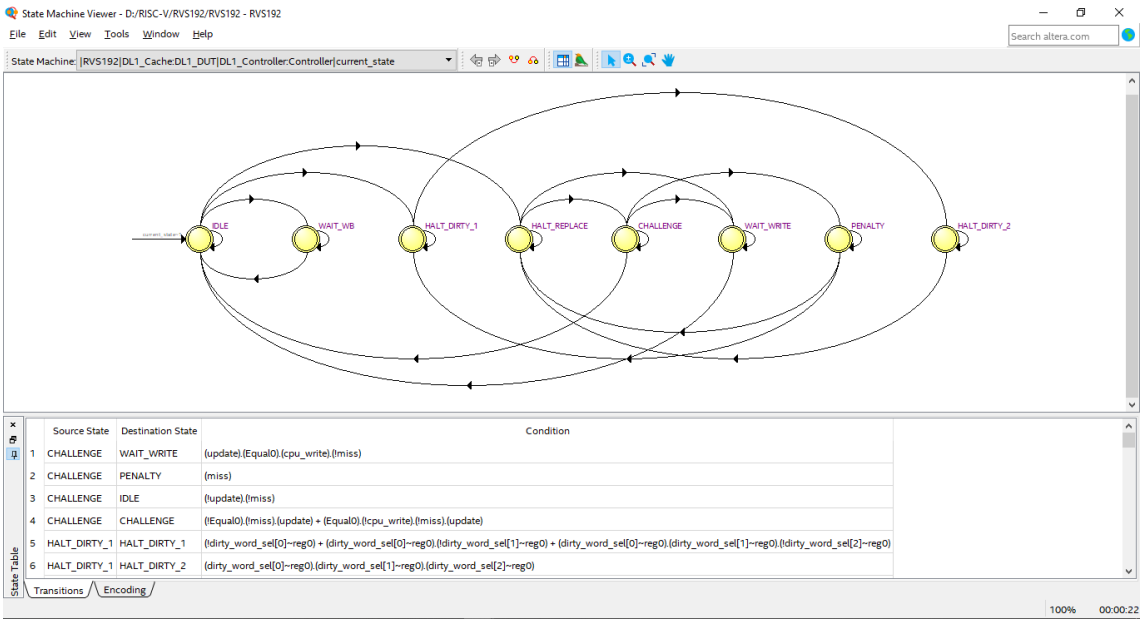
Tương tự IL1 Controller, DL1 Controller cũng gồm 2 bộ FSM riêng biệt:

- DL1 inclusion FSM: đây là FSM dùng để đảm bảo tính chất inclusive của DL1 Cache. FSM này sẽ liên tục chờ tín hiệu kích hoạt từ L2 Cache. Nếu như có yêu cầu discard một line nhớ trong L2 Cache, tín hiệu yêu cầu *inst_replace_req* hoặc *data_replace_req* được gửi tới DL1 Cache. Lúc này DL1 Cache sẽ kiểm tra line nhớ đó có thuộc sở hữu của mình không. Nếu có và line đó không dirty, DL1 inclusion FSM sẽ loại bỏ (discard) line nhớ, sau đó gửi *inst_replace_dl1_ack* để báo cho L2 Cache. Nếu line đó có và dirty thì yêu cầu gửi line cần loại bỏ xuống L2 Cache, rồi thực hiện tương tự như trên. Nếu không, nó chỉ gửi *inst_replace_dl1_ack*. Trường hợp có 2 yêu cầu discard cùng lúc do L2 Cache yêu cầu thay thế line nhớ (để lưu cả inst và data mới), FSM này sẽ xử lý tuần tự inst rồi đến data.



Hình 38: DL1 inclusion FSM

- DL1 FSM: đây là FSM sẽ quản lý việc cập nhập dữ liệu mới cho DL1 Cache. Khi có *miss_write* xảy ra, nghĩa là dữ liệu cần thay thế không có trong Write Buffer lẫn DL1 Cache, đồng thời Write Buffer đang đầy. DL1 FSM sẽ vào trạng thái WAIT_WB, chờ tín hiệu wb_full hết tích cực. Trong trường hợp *miss_read*, DL1 FSM kiểm tra xem line cần thay thế có bị dirty không. Nếu có, DL1 FSM vào trạng thái HALT_DIRTY_1, chờ gửi hết word trong line bị dirty thì chuyển qua HALT_DIRTY_2 chờ ack từ L2 Cache. Sau đó, quá trình cập nhập sẽ diễn ra tương tự IL1 FSM.

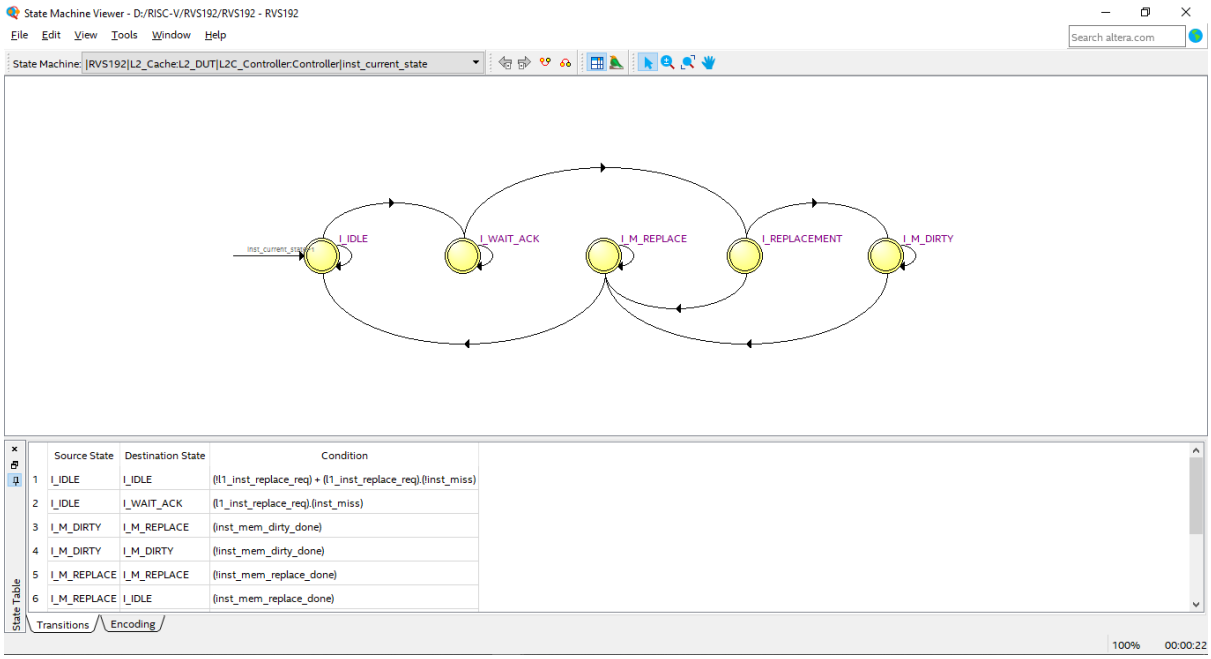


Hình 39: DL1 FSM

8.2.1.3 L2 Controller

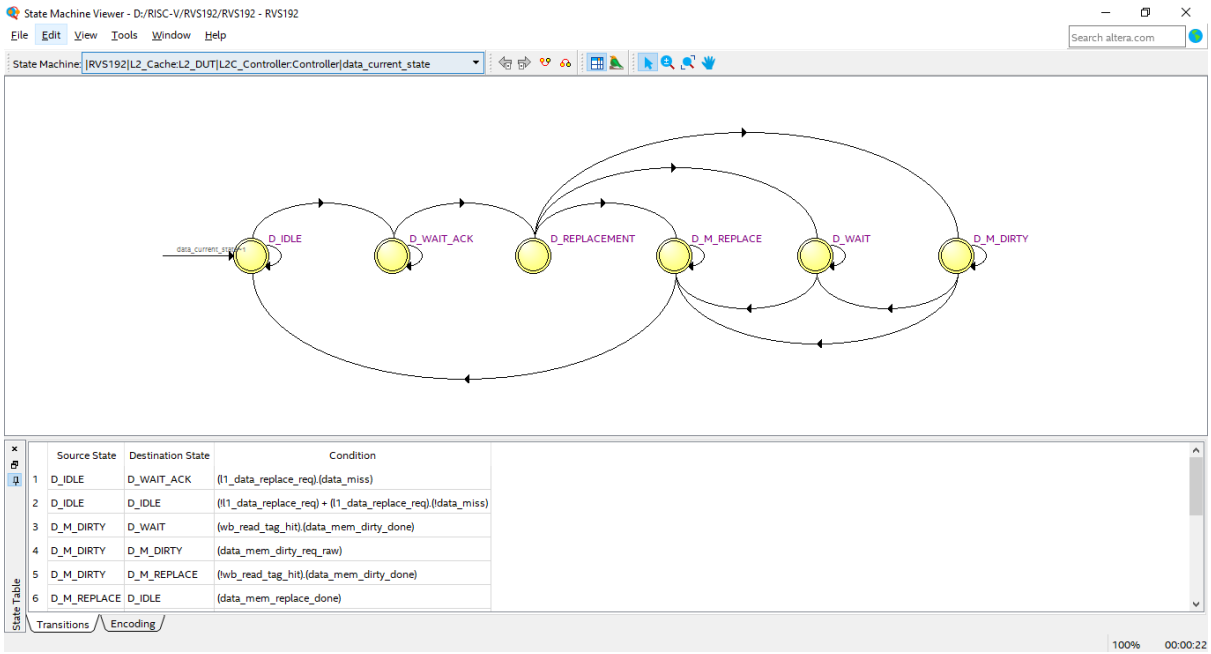
Để có thể xử lý song song cả miss_inst và miss_data. L2 Controller sẽ tích hợp 2 bộ FSM riêng biệt cho inst và data:

- L2 INST FSM: FSM quản lý cho miss từ IL1 Cache. Sau khi có yêu cầu cập nhập từ IL1 Cache. L2 INST FSM kiểm tra line dữ liệu cần đưa lên IL1 Cache có trong L2 Share Cache hay không, nếu có thì giữ trạng thái I_IDLE và đưa trực tiếp lên IL1 Cache (2 miss trùng nhau không bao giờ xảy ra ở L2 Cache). Nếu line cần tìm không có trong L2 Share Cache, L2 INST FSM gửi yêu cầu kiểm tra line được thay thế lên IL1 Cache và chuyển sang trạng thái I_WAIT_ACK, chờ inst_replace_il1_ack_sync hoặc inst_replace_dl1_ack_sync tích cực. L2 INST FSM kiểm tra tiếp liệu line cần thay thế trong L2 Share Cache có dirty không. Nếu dirty thì nhảy sang I_M_DIRTY và yêu cầu gửi line dirty xuống tầng Memory thấp hơn. Tiếp theo, L2 INST FSM nhảy tới I_M_REPLACE, yêu cầu dữ liệu từ Memory. Sau quá trình này L2 INST FSM sẽ về I_IDLE.



Hình 40: L2 INST FSM

- L2 DATA FSM: FSM quản lý cho miss từ DL1 Cache. Ở 2 trạng thái đầu là D_IDLE, D_WAIT_ACK và D_REPLACEMENT, L2 DATA FSM hoạt động tương tự như L2 INST FSM. Tuy nhiên, các trạng thái khác sẽ có thay đổi. Cụ thể, ở trạng thái D_M_DIRTY sau khi gửi xong line bị dirty xuống Memory, L2 DATA FSM phải kiểm tra xem có word nào thuộc line cần thay thế đang ở trong L2 Write Buffer hay không. Nếu có thì phải chờ word đó được đưa xuống Memory. Sau đó L2 DATA FSM sẽ vào D_M_REPLACE và hoạt động tương tự như L2 INST FSM.

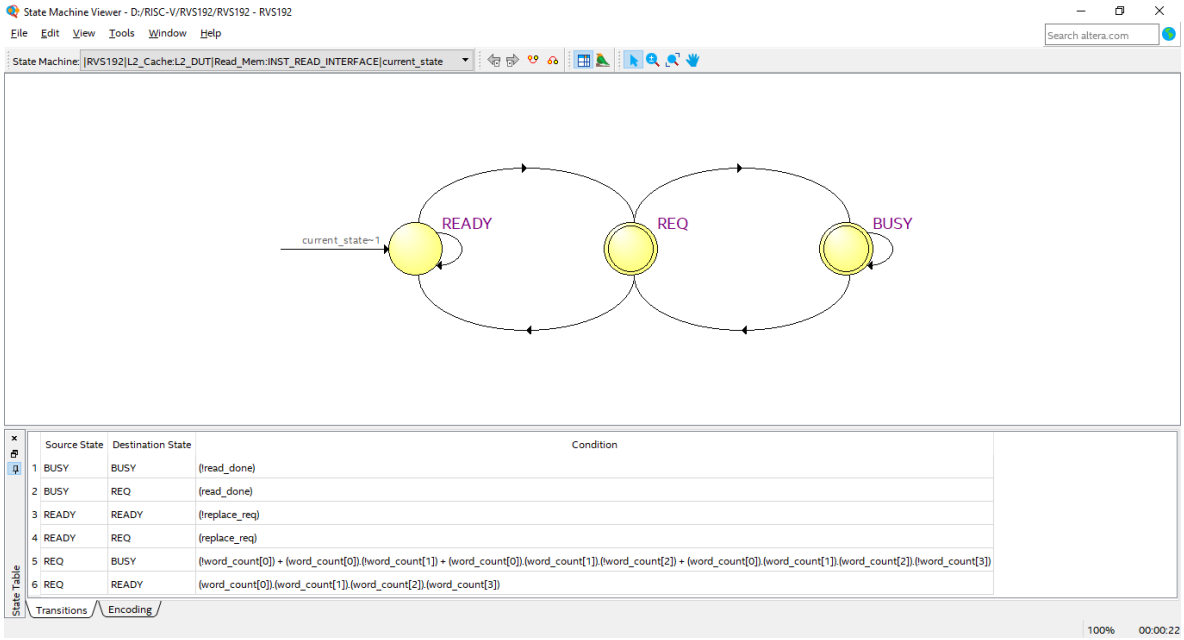


Hình 41: L2 DATA FSM

8.2.1.4 Mem Interface Controller

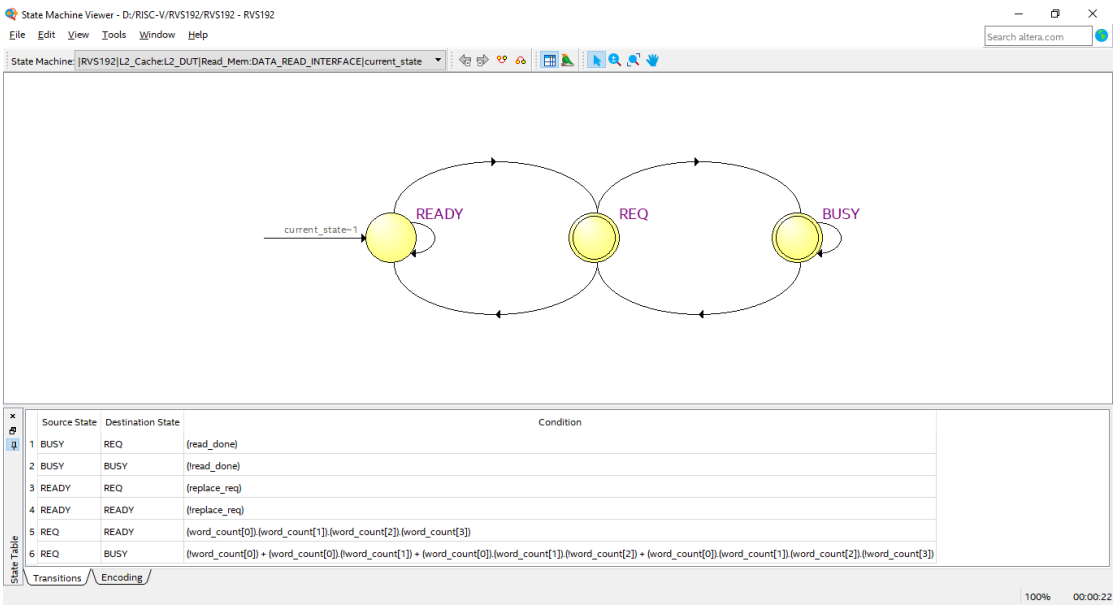
Do RVS192 hỗ trợ cấu trúc Harvard, phần giao tiếp với Memory yêu cầu phải có 3 FSM:

- INST READ FSM: FSM dùng cho việc đọc bộ nhớ lệnh.



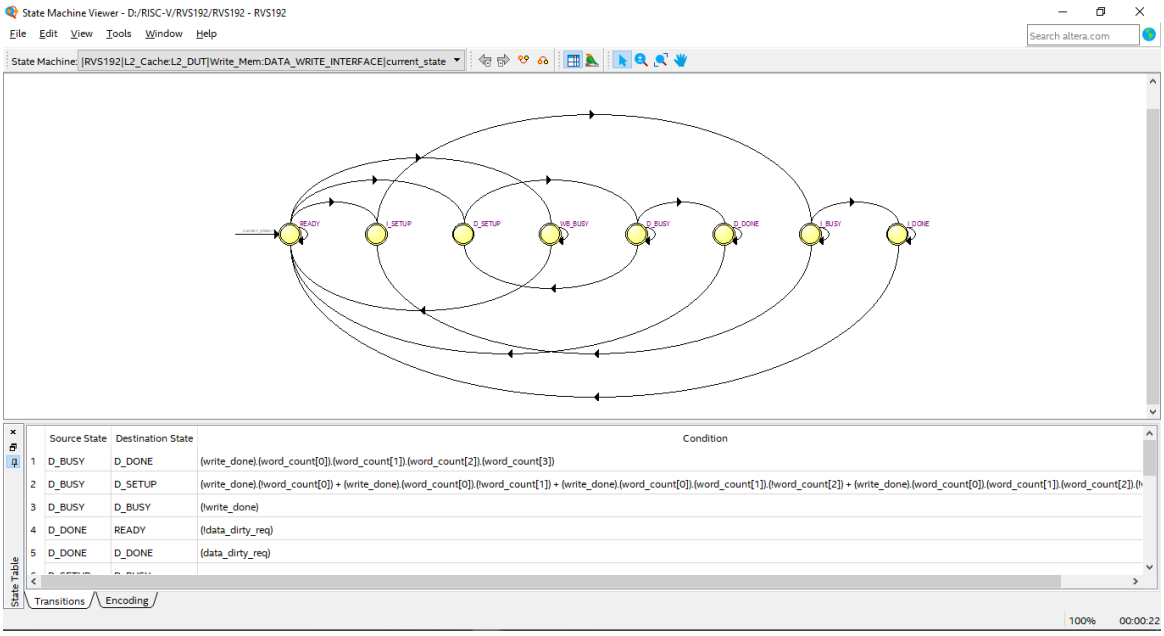
Hình 42: INST READ FSM

- DATA READ FSM: FSM dùng cho việc đọc bộ nhớ dữ liệu.



Hình 43: DATA READ FSM

- DATA WRITE FSM: FSM dùng cho việc ghi bộ nhớ dữ liệu. Các trạng thái I_SETUP, I_BUSY và I_DONE sẽ xử lý việc ghi dirty line từ L2 INST FSM. Tương tự, các trạng thái D_SETUP, D_BUSY và D_DONE sẽ xử lý việc ghi dirty line từ L2 DATA FSM. WB_BUSY sẽ xử lý việc ghi từ L2 Write Buffer. Nếu như ở trạng thái READY, 3 yêu cầu ghi cùng lúc được kích hoạt. Thứ tự thực hiện: INST => DATA => WB.



Hình 44: DATA WRITE FSM

8.2.2 Hiện thực Pipeline RISC-V

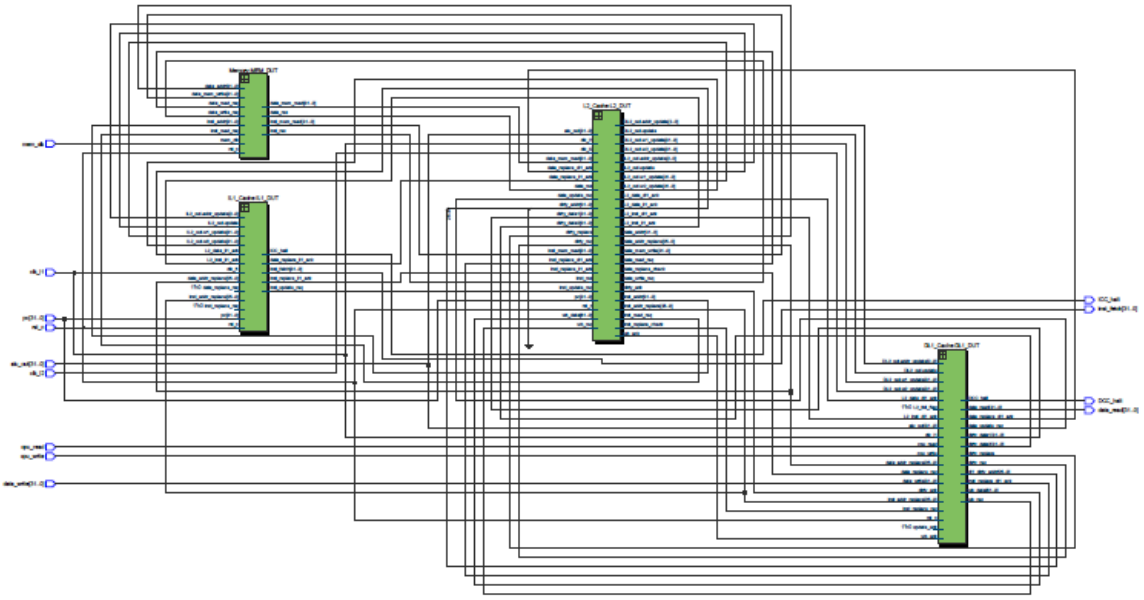
Từ nền tảng môn học cấu trúc máy tính ở học kỳ 191, đồ án học kỳ 192 kế thừa các ưu điểm của 6 Stage Pipeline RISC-V. Từ đó, RVS192 được tối ưu hóa để sử dụng hiệu quả các nguồn tài nguyên. Toàn bộ source code đều được chuyển sang sử dụng SystemVerilog cho hiệu quả khi thiết kế cao hơn. Các thay đổi của Pipeline RISC-V (các thay đổi này được xét khi RISC-V chưa được thêm hệ thống Cache):

- Chuyển về 5 tầng: giảm cycle sửa lỗi lệnh nhảy.
- Các tín hiệu điều khiển được packed để dễ dàng quản lý.
- Bộ Controller được chuyển thành bộ Decoder với cách quản lý tín hiệu hiệu quả hơn.
- Tầng Execute được thay đổi: bỏ bộ Branch Compare và Branch Check thay bằng bộ Branch Handling. ALU được thiết kế lại để tối ưu tài nguyên sử dụng.
- Tầng Fetch có những thay đổi ở việc chọn lựa pc và pc reset.
- Các bộ Updata FSM được tích hợp vào bộ Branch Handling giúp giảm tài nguyên sử dụng và các tín hiệu cần phải truyền giữa các tầng Pipeline.

8.3 Quy trình kiểm định

8.3.1 Kiểm tra chức năng và phần cứng tổng hợp của Cache.

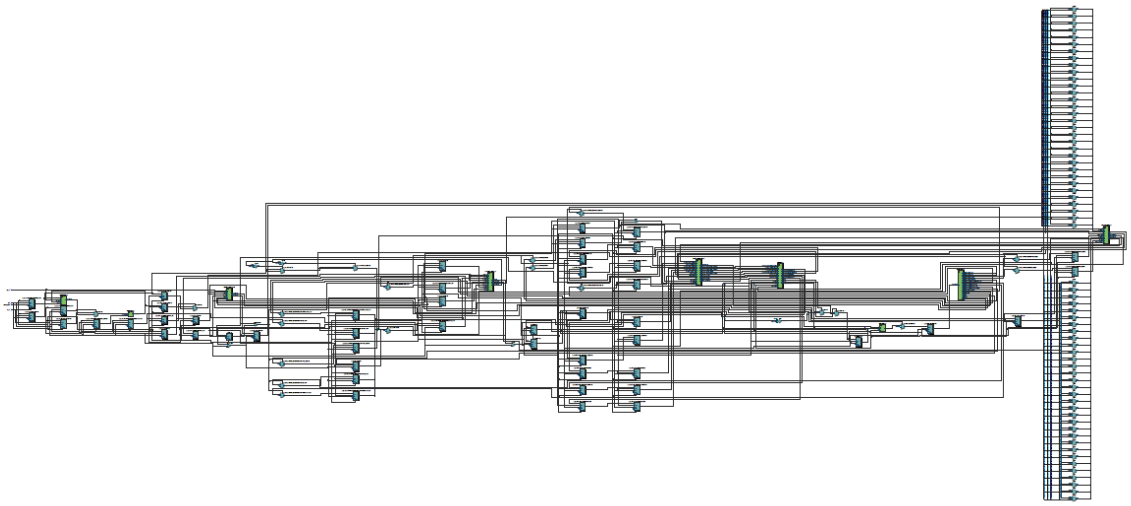
- Lần lượt kiểm tra phần cứng tổng hợp của IL1 Cache, DL1 Cache và L2 Cache.
- Ghép các level Cache với nhau thành một hệ thống hoàn chỉnh (chưa được gắn vào pipeline), sau đó xây dựng một trường và các testcase để kiểm định chức năng của Cache.



Hình 45: Cache System

8.3.2 Kiểm tra chức năng và phần cứng tổng hợp của Pipeline 5 tầng.

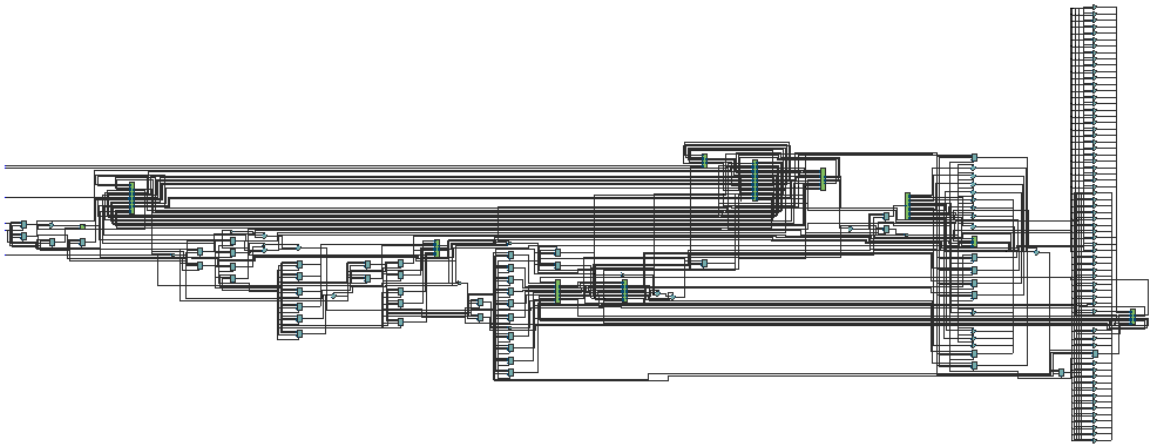
- Các bài test tính Factorial và Arrangement sẽ được sử dụng lại.



Hình 46: 5 Stage Pipeline, no Cache, Branch Predictor

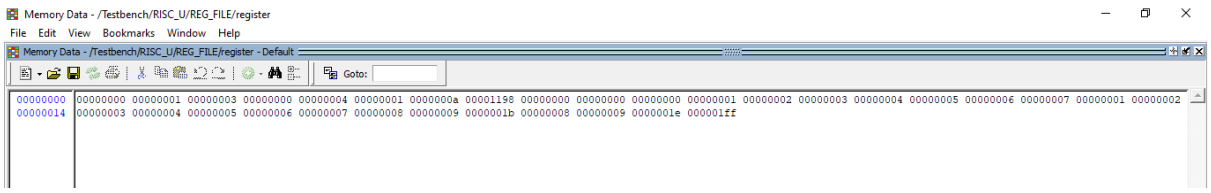
8.3.3 Kiểm tra chức năng và phần cứng tổng hợp của RVS192.

- Các bài test tính Factorial và Arrangement sẽ được sử dụng lại.
- Arrangement sẽ được thêm phần đọc lại giá trị từ bộ nhớ để kiểm tra tính đúng đắng của pipeline khi tích hợp thêm Cache.

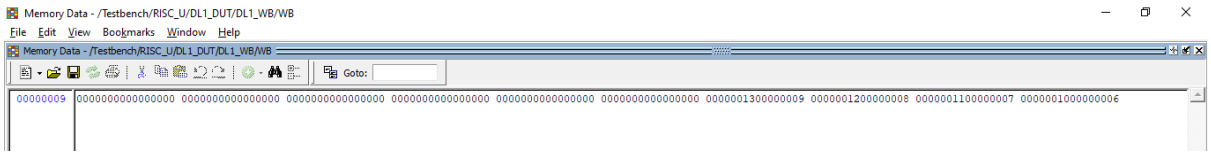


Hình 47: 5 Stage Pipeline, Cache, Branch Predictor

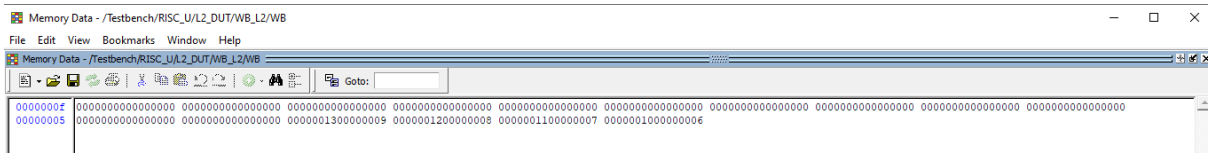
- Phân tích bài test *arrangement_cache_test1: đọc 10 word dữ liệu trong memory ở vị trí 0x0-0xa sau đó sắp xếp từ nhỏ đến lớn, dữ liệu sau khi sắp xếp nằm trong các thanh ghi s1-s10, lưu các thanh ghi này vào các ô nhớ 0x28-0x4c memory, cuối cùng đọc ngược dữ liệu từ các ô nhớ vừa lưu vào các thanh ghi a0-a7 và t3-t4.*



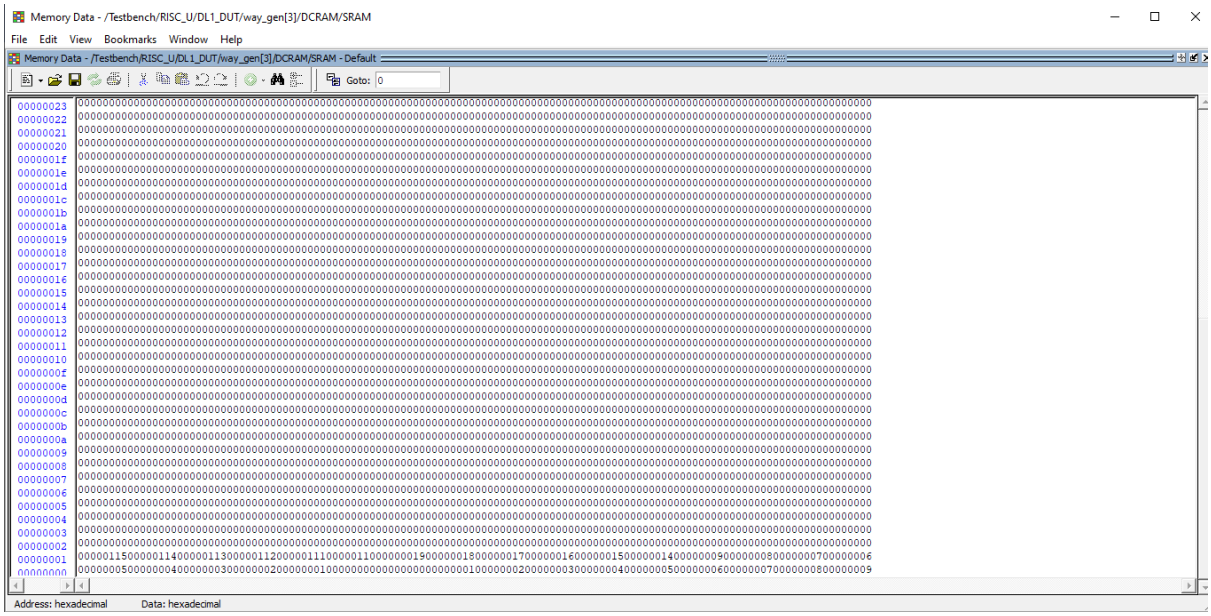
Hình 48: Register File – Arrangement cache test1



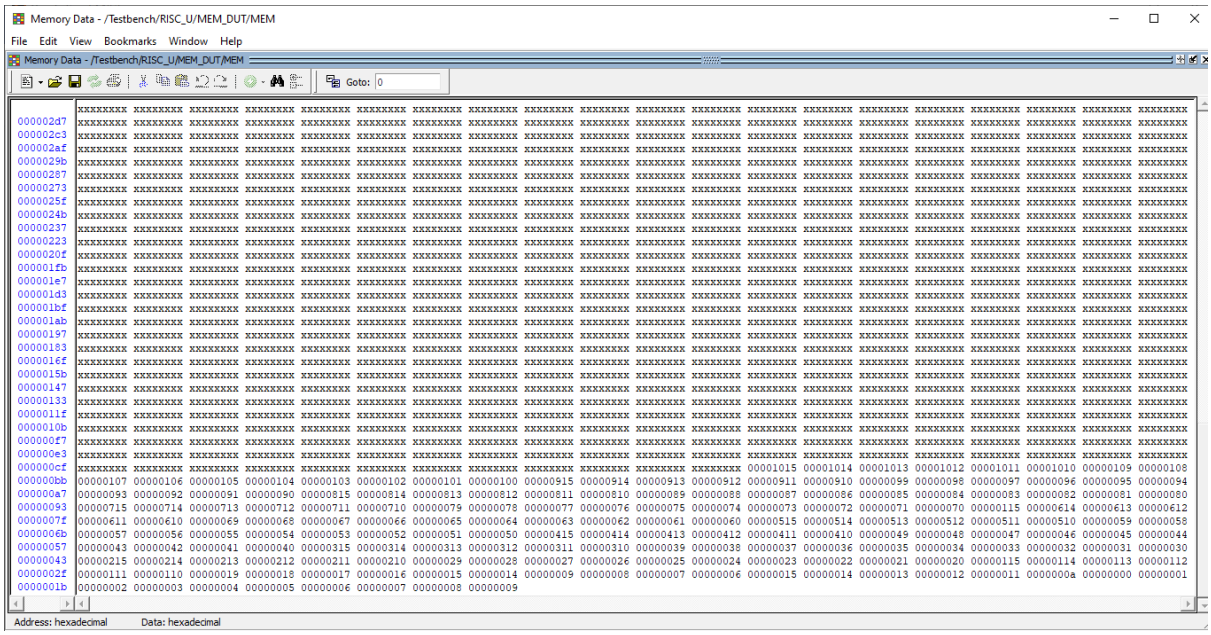
Hình 49: DL1 Write Buffer - Arrangement cache test1



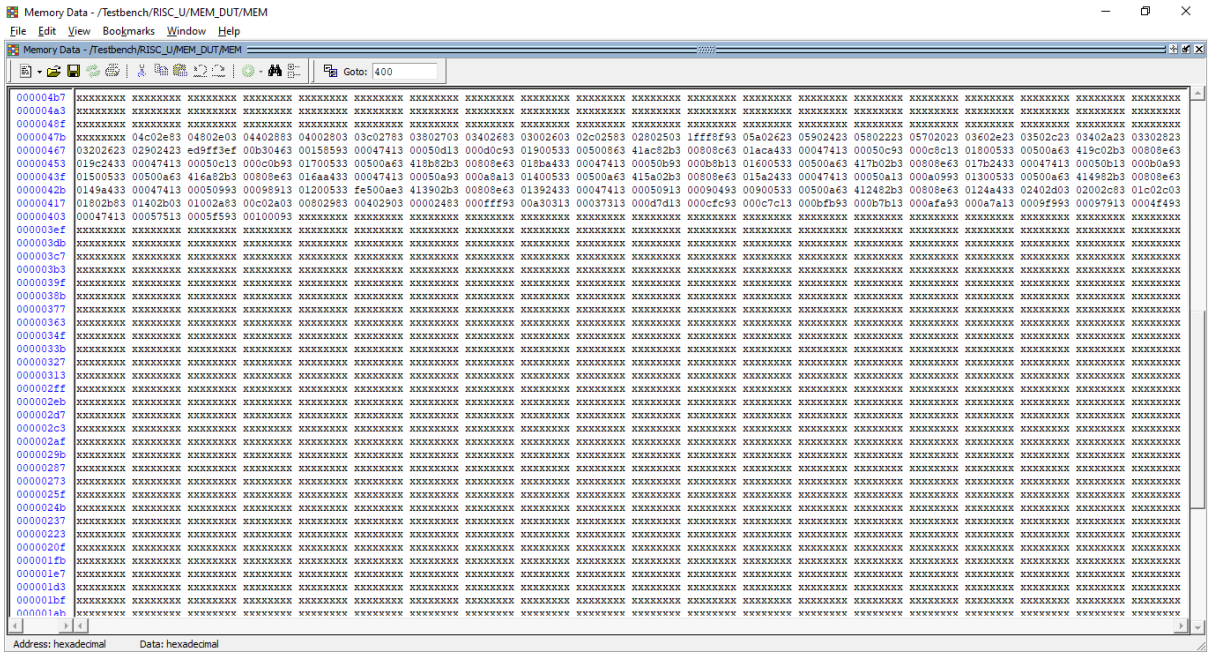
Hình 50: L2 Write Buffer - Arrangement cache test1



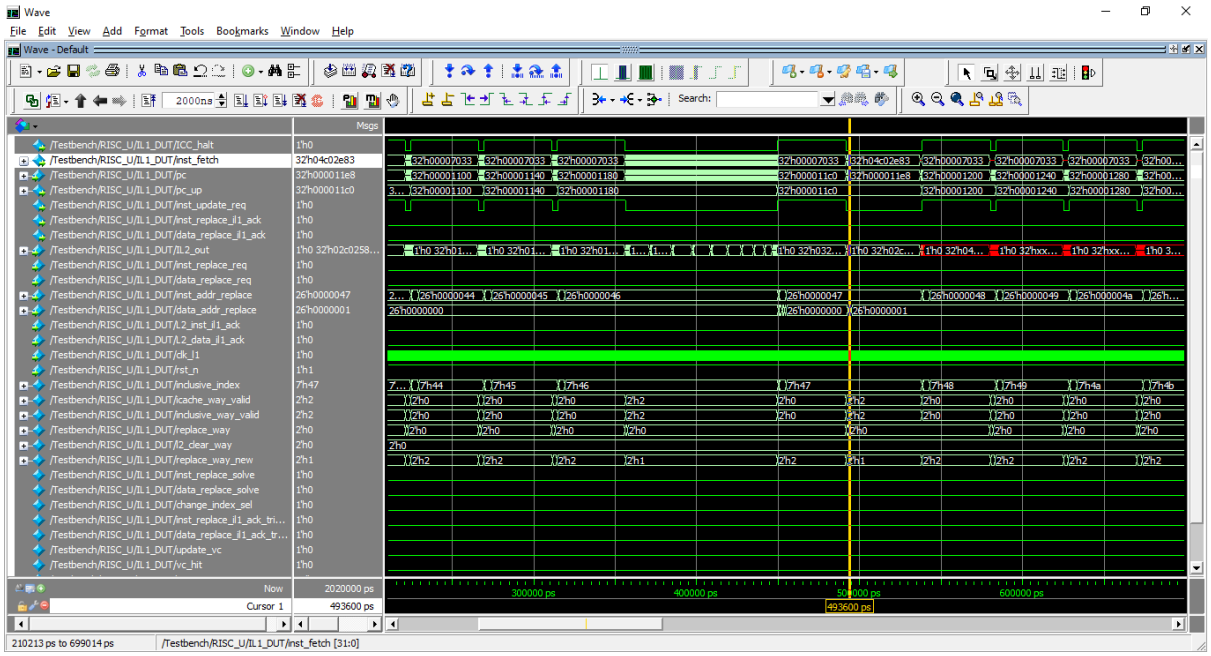
Hình 51: DL1 Cache – DATA SRAM – way 3 - Arrangement cache test1



Hình 52: Data Memory- Arrangement cache test1



Hình 53: Inst Memory- Arrangement cache test1



Hình 54: Waveform - Arrangement cache test1

Từ Waveform ta có thể thấy sau một khoảng thời gian setup để Cache load tất cả các lệnh trong Inst Memory lên IL2 Cache, Cache xác lập và hit liên tục. Lệnh cuối cùng cần được thực hiện là 32'h04c02e83 (hình 52) sẽ kết thúc chương trình. Trong cấu hình mô phỏng, tốc độ truy cập Level 1 Cache gấp đôi Level 2 Cache và gấp 3 Memory, một khi Cache đã xác lập, tốc độ xử lý chắc chắn sẽ gấp 3 lần lúc chưa tích hợp Cache.

Vì một block dữ liệu chứa 16 words, khi load block chứa dữ liệu cần sắp xếp lên và ghi ngược dữ liệu sau sắp xếp lại. Chỉ có 6 words địa chỉ thấp được lưu vào DL1 Cache (hình 50), tương tự cho L2 Cache. 4 words cao lúc này sẽ được lưu vào DL1 Write Buffer (48) sau đó đưa xuống L2 Write Buffer (49) rồi cuối cùng lưu vào Memory (51).

9. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

9.1 Kết luận

Sau khi thực hiện đề tài, em đã rút ra một số kinh nghiệm trong việc mô tả RTL code, các cấu trúc RAM khác nhau, các đặc tính hỗ trợ trong một hệ thống Cache, cách để thiết kế datapath từ một tập lệnh cho trước và các cách đồng bộ dữ liệu giữa các miền clock.

- Ưu điểm: Hoàn thành được cấu trúc RVS192 theo những tiêu chí đã đặt ra và vượt qua được các bài test cơ bản. Mô tả RTL code hiệu quả giúp rút ngắn thời gian debug. Đồng thời việc sửa lỗi cũng sẽ đơn giản và dễ dàng hơn.
- Nhược điểm: các bài test chức năng của RVS192 vẫn chưa bao phủ hết các trường hợp có thể gây ra lỗi ở Cache. Ngoài ra, RVS192 vẫn chưa được đổ lên FPGA và các chỉ số hiệu suất vẫn chưa được kiểm định.

9.2 Hướng phát triển

Xây dựng môi trường test cpu để bao phủ các trường hợp cần kiểm tra.

Ở các dự án tiếp theo, RVS192 sẽ được kế thừa để phát triển lên thành một hệ thống SoC có các thành phần như bus hệ thống và các ngoại vi giao tiếp.

10. TÀI LIỆU THAM KHẢO

University of California, Berkeley, “The RISC-V Instruction Set Manual”

The university of Edinburgh, “Cache Performance”

Altera Corp., “Recommended HDL Coding Styles”

https://en.wikipedia.org/wiki/Cache_inclusion_policy

11.PHỤ LỤC

Source code của RVS192 được pubic trên trang Github của tác giả: hungbk99