# Giới thiệu đề tài

## Tổng quan đề tài

Mục tiêu của việc thiết kế vi xử lý là đạt được hiệu quả và tốc độ cao, nhưng quá trình đọc và ghi từ vi xử lý tới bộ nhớ chiếm một lượng lớn thời gian, do đó ảnh hưởng rất lớn đến tốc độ của vi xử lý. Sử dụng bộ nhớ đệm, hay được gọi là cache, chính là giải pháp giúp tối ưu được tốc độ cho vi xử lý.

Cache là một bộ nhớ nhỏ có tốc độ truy xuất cao và được thiết kế gần vi xử lý (có thể ở trong hoặc ngoài tùy thuộc vào level của cache và yêu cầu thiết kế). Cache chứa dữ liệu thường xuyên được truy cập từ bộ nhớ chính, qua đó giúp giảm thời gian truy cập dữ liệu của vi xử lý. Cache thường được thiết kế nhiều tầng và tạo thành cấu trúc theo tầng gọi là memory hierarchy, việc thiết kế nhiều tầng cache (multilevel caches) giúp cải thiện hiệu năng của cache nhờ việc giảm thời gian dữ liệu được truyền tải từ bộ nhớ chính khi cache không chứa dữ liệu cần truy cập.

## 1.2. Mục tiêu đề tài

Mục tiêu của đề tài là sử dụng ngôn ngữ mô tả phần cứng SystemVerilog thiết kế 5-stage pipelined RISC-V processor dựa trên tập lệnh RV32I, processor được xử lý hazard bằng kỹ thuật forwarding, có bộ branch prediction 2-bit scheme. Thiết kế bộ nhớ cho processor bao gồm 2 tầng, L1 và L2 cache. Trong đó có 2 L1 caches, bao gồm 1 data cache và 1 instruction cache, 1 L2 cache có kích thước lớn hơn nhưng chậm hơn. Cache được thiết kế theo cấu trúc set associative có sử dụng giải thuật tree-PLRU cho replacement policy.

## 1.3. Giới hạn đề tài

## 1.4. Cấu trúc của báo cáo

## 1.5. Source code

# Cơ sở lý thuyết

## 2.1. Giới thiệu về RISC-V

RISC-V là một kiến trúc tập lệnh (ISA - Instruction Set Architecture) theo chuẩn RISC (Reduced Instruction Set Computer), được thiết kế với mục tiêu đơn giản, linh hoạt và mở. Khác với nhiều ISA thương mại như x86 hay ARM, RISC-V có mã nguồn mở và được phát triển nhằm phục vụ cả giáo dục lẫn thương mại. Các đặc điểm nổi bật của RISC-V đã làm cho nó trở thành một sự lựa chọn hấp dẫn trong nhiều lĩnh vực từ thiết kế hệ thống nhúng cho đến các ứng dụng máy tính cao cấp.

RISC-V bắt đầu vào năm 2010 tại Đại học California, Berkeley, bởi một nhóm nghiên cứu do giáo sư David Patterson và các đồng nghiệp dẫn đầu. Nhóm nghiên cứu này đặt mục tiêu tạo ra một kiến trúc đơn giản, dễ học và dễ triển khai. Một trong những lý do chính để phát triển RISC-V là để giải quyết những hạn chế của các kiến trúc tập lệnh hiện có, chẳng hạn như sự phức tạp, chi phí cấp phép cao, và không thể mở rộng tự do. RISC-V được thiết kế với tư duy mở, có thể sử dụng trong cả giáo dục lẫn thương mại mà không gặp các vấn đề bản quyền.

RISC-V tuân theo các nguyên tắc chính của kiến trúc RISC, bao gồm:

* Đơn giản hóa tập lệnh: Chỉ sử dụng một tập nhỏ các lệnh cơ bản, mỗi lệnh có độ dài cố định và đơn giản. Điều này giúp tối ưu hóa quá trình xử lý, giảm thiểu số lệnh phức tạp cần phải xử lý.
* Load-Store Architecture: RISC-V là kiến trúc load-store, tức là tất cả các thao tác trên dữ liệu đều được thực hiện trong thanh ghi, còn việc truy cập bộ nhớ chỉ được thực hiện thông qua các lệnh load và store.
* Độ dài lệnh cố định: Các lệnh trong RISC-V có độ dài 32-bit, giúp đơn giản hóa bộ giải mã và tăng hiệu quả khi xử lý.
* Modular: RISC-V được thiết kế với khả năng mở rộng, cho phép người dùng thêm các phần mở rộng cho các ứng dụng đặc thù như tính toán số nguyên, số chấm động, hoặc các ứng dụng vector.

Các đặc điểm nổi bật của RISC-V:

* Mã nguồn mở: RISC-V hoàn toàn miễn phí và mã nguồn mở, giúp các nhà phát triển có thể sử dụng và tùy chỉnh mà không phải trả phí cấp phép như với các kiến trúc độc quyền khác.
* Tính mô-đun và linh hoạt: RISC-V cho phép người dùng thêm hoặc bớt các phần mở rộng theo nhu cầu cụ thể của từng hệ thống. Ví dụ, có thể thêm phần mở rộng xử lý dấu phẩy động (F), phần mở rộng vector (V), hoặc phần mở rộng bảo mật (S).
* Khả năng mở rộng: Kiến trúc RISC-V có thể được triển khai cho các hệ thống từ nhúng (32-bit) đến các siêu máy tính (64-bit hoặc 128-bit). Điều này cho phép RISC-V đáp ứng được nhiều yêu cầu và mức độ phức tạp của các ứng dụng khác nhau.
* Hiệu suất cao và tiết kiệm năng lượng: Với việc tuân thủ nguyên lý thiết kế RISC, RISC-V cho phép tối ưu hóa hiệu suất tính toán và tiêu thụ năng lượng, đặc biệt phù hợp với các ứng dụng di động và nhúng.

RISC-V được chia thành nhiều phiên bản để phù hợp với các nhu cầu và ứng dụng khác nhau:

* RV32: Kiến trúc 32-bit, phù hợp với các ứng dụng nhúng, hệ thống nhỏ gọn.
* RV64: Kiến trúc 64-bit, phù hợp với các ứng dụng yêu cầu tính toán cao hoặc lượng bộ nhớ lớn.
* RV128: Kiến trúc 128-bit, được thiết kế cho các ứng dụng tương lai với yêu cầu rất lớn về hiệu năng và không gian địa chỉ.

## 2.2. Kiến trúc tập lệnh RV32I

Kiến trúc tập lệnh RV32I là tập lệnh cơ bản của RISC-V, được thiết kế để hoạt động trên bộ vi xử lý với không gian địa chỉ 32-bit. RV32I là tập lệnh cơ bản và nhỏ gọn, hỗ trợ các thao tác xử lý đơn giản và là nền tảng cho các tập lệnh mở rộng khác của RISC-V (như RV64I cho kiến trúc 64-bit). Dưới đây là các khái niệm và đặc điểm chính của RV32I:

a. Đặc điểm cơ bản của RV32I:

* 32-bit word: RV32I sử dụng word 32-bit cho địa chỉ và dữ liệu.
* Tối ưu hóa cho đơn giản: Tập lệnh của RV32I được thiết kế đơn giản, với số lượng ít các loại lệnh cơ bản, giúp dễ dàng triển khai và tối ưu hóa.
* Tính mô-đun: RV32I là một tập lệnh cơ sở, có thể được mở rộng với các tập lệnh khác như RV32E, RV32M (nhân chia), hoặc các phần mở rộng khác như F, D (số dấu phẩy động).

b. Các nhóm lệnh chính của RV32I:

* Lệnh Load và Store: các lệnh load (tải dữ liệu từ bộ nhớ vào thanh ghi) và store (lưu dữ liệu từ thanh ghi vào bộ nhớ) là cơ bản trong mọi kiến trúc RISC. Ví dụ: LW (Load Word), SW (Store Word), LB (Load Byte), SB (Store Byte).
* Lệnh tính toán số nguyên: các lệnh này thực hiện các phép tính toán số nguyên giữa các thanh ghi. Ví dụ: ADD (cộng), SUB (trừ), AND, OR, XOR.
* Lệnh dịch và so sánh: dịch bit (shift) và so sánh là các thao tác phổ biến trong nhiều ứng dụng. Ví dụ: SLL (dịch trái), SRL (dịch phải không dấu), SLT (so sánh nhỏ hơn).
* Lệnh nhảy và phân nhánh: các lệnh này điều khiển luồng chương trình, cho phép nhảy đến các địa chỉ khác nhau trong bộ nhớ. Ví dụ: JAL (Jump and Link), BEQ (nhảy khi bằng), BNE (nhảy khi không bằng).
* Lệnh immediate: đây là các lệnh tính toán với một hằng số (immediate) được gán trực tiếp vào lệnh. Ví dụ: ADDI (cộng với immediate), ORI (OR với immediate).
* Lệnh hệ thống: những lệnh này liên quan đến tương tác với phần mềm và hệ thống, như gọi hệ thống hoặc quản lý ngoại lệ. Ví dụ: ECALL (gọi hệ thống), EBREAK (dừng chương trình).

c. Thanh ghi và các đặc điểm liên quan:

* 32 thanh ghi: RV32I có 32 thanh ghi toàn phần (x0 - x31), mỗi thanh ghi rộng 32-bit. Trong đó:
* x0 luôn có giá trị bằng 0.
* Các thanh ghi khác có thể lưu trữ dữ liệu tạm thời và giá trị địa chỉ.
* Program Counter (PC): Thanh ghi PC lưu trữ địa chỉ của lệnh hiện tại đang được thực thi.

d. Định dạng lệnh của RV32I:

Tất cả các lệnh trong RV32I đều có độ dài cố định là 32 bit. Có năm định dạng lệnh chính trong RV32I, được chia theo các trường khác nhau:

* R-type: Dùng cho các lệnh tính toán số nguyên giữa các thanh ghi.
* I-type: Dùng cho các lệnh tính toán có immediate, cũng như các lệnh load.
* S-type: Dùng cho các lệnh store.
* B-type: Dùng cho các lệnh nhảy có điều kiện.
* J-type: Dùng cho các lệnh nhảy không điều kiện.

e. Ưu điểm của RV32I:

* Đơn giản và linh hoạt: RV32I chỉ có các lệnh cơ bản, giúp việc triển khai phần cứng trở nên dễ dàng và nhanh chóng.
* Tối ưu hóa hiệu suất và điện năng: Kiến trúc đơn giản giúp tiết kiệm tài nguyên phần cứng và giảm mức tiêu thụ điện năng.
* Tính mở rộng: RV32I có thể được mở rộng với các phần mở rộng khác nhau tùy theo nhu cầu ứng dụng.

f. Ứng dụng của RV32I:

RV32I chủ yếu được sử dụng trong các hệ thống nhúng, các thiết bị có giới hạn về tài nguyên và yêu cầu tiết kiệm năng lượng, nhưng cũng có thể mở rộng để đáp ứng các yêu cầu phức tạp hơn.

## 2.3. Pipeline

## 2.4. Branch prediction

## 2.5. Cache

### 2.5.1. Khái niệm

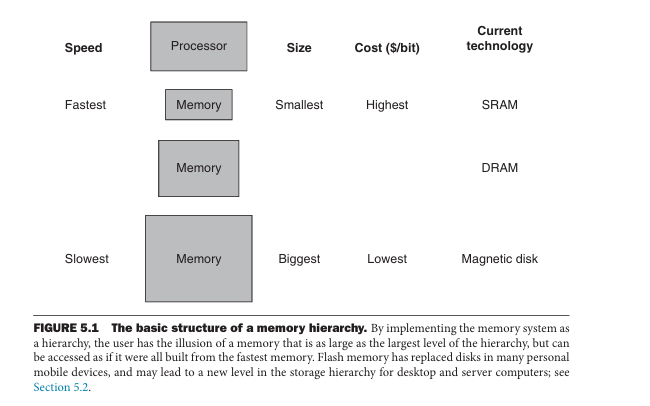
Bộ nhớ đệm, được gọi là cache, nó là vùng bộ nhớ có tốc độ truy xuất cao dùng để lưu trữ tạm thời những câu lệnh chương trình hoặc dữ liệu. Những câu lệnh và dữ liệu này được trích xuất từ bộ nhớ chính và thường được sử dụng lại ngay sau đó. Mục đích chính của bộ nhớ cache là tăng tốc độ truy cập lặp lại đối với cùng một địa chỉ bộ nhớ và những địa chỉ ở gần đó. Để đạt được hiệu quả, truy cập dữ liệu đối với cache phải nhanh hơn rất nhiều so với truy cập vào dữ liệu ở bộ nhớ chính.

Khi cache được sử dụng, mỗi lần truy cập dữ liệu bộ nhớ đều bắt đầu tìm kiếm từ cache. Nếu dữ liệu yêu cầu có trong cache, processor sẽ trích xuất và sử dụng nó ngày lập tức, đây được gọi là cache hit. Nếu dữ liệu không tìm kiếm được trong cache, đây được gọi là cache miss, instruction hay data phải được trích xuất từ bộ nhớ. Trong quá trình lấy dữ liệu yêu cầu, nội dung được sao chép vào cache để sử dụng sau đó.

Bộ nhớ cache cải thiện hiệu suất của máy tính vì nhiều thuật toán được hệ điều hành và các ứng dụng thực thi thể hiện tính cục bộ tham chiếu. Tính cục bộ tham chiếu đề cập đến việc tái sử dụng dữ liệu đã được truy cập gần đây (điều này được gọi là tính cục bộ theo thời gian) và việc truy cập dữ liệu ở gần với dữ liệu đã được truy cập trước đó (điều này được gọi là tính cục bộ theo không gian).

Thông thường, các vùng bộ nhớ cache có kích thước nhỏ hơn so với bộ nhớ chính. Cache được thiết kế để đạt tốc độ tối đa, điều này có nghĩa là chúng phức tạp hơn và tốn kém hơn tính theo mỗi bit so với công nghệ lưu trữ dữ liệu được sử dụng trong bộ nhớ chính. Do kích thước hạn chế, cache thường nhanh chóng bị lấp đầy. Khi bộ nhớ cache không còn trống để lưu trữ dữ mới, một dữ liệu cũ phải bị loại bỏ. Cache controller sẽ sử dụng một chính sách thay thế để chọn dữ liệu sẽ bị ghi đè bởi dữ liệu mới.

Mục tiêu của cache trong processor là tối đa hóa tỷ lệ cache hit theo thời gian, từ đó thực thi được những câu lệnh với tốc độ cao nhất có thể. Để đạt được mục tiêu này, cache phải xác định những câu lệnh và dữ liệu nào sẽ được đặt vào cache và giữ lại để sử dụng trong tương lai.

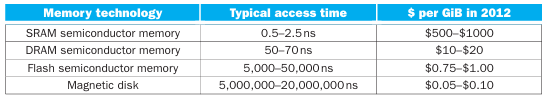
****

Hình . Cấu trúc cơ bản của hệ thống memory

Hình trên mô tả hệ thống bộ nhớ theo các tiêu chí như tốc độ, kích thước, chi phí và công nghệ. Đối với các loại bộ nhớ được thiết kế càng gần processor thì tốc độ phải càng tăng và kích thước càng nhỏ, tương ứng với đó thì chi phí cũng càng tăng.

### 2.5.2. Công nghệ bộ nhớ

Có bốn công nghệ chính được sử dụng trong hệ thống phân cấp bộ nhớ hiện nay. Bộ nhớ chính được triển khai bằng DRAM (dynamic random access memory), trong khi các cấp gần bộ xử lý hơn (bộ nhớ cache) sử dụng SRAM (static random access memory). DRAM có chi phí trên mỗi bit thấp hơn so với SRAM, mặc dù nó chậm hơn đáng kể. Sự chênh lệch về giá xuất phát từ việc DRAM sử dụng diện tích ít hơn đáng kể trên mỗi bit bộ nhớ, do đó DRAM có dung lượng lớn hơn với cùng số lượng silicon. Công nghệ thứ ba là bộ nhớ flash. Đây là bộ nhớ không bay hơi (nonvolatile) được sử dụng làm bộ nhớ thứ cấp trong các thiết bị di động cá nhân. Công nghệ thứ tư, được sử dụng để triển khai có kích thước lớn nhất và chậm nhất trong hệ thống phân cấp bộ nhớ trong các máy chủ là đĩa từ. Thời gian truy cập và chi phí trên mỗi bit khác nhau đáng kể giữa các công nghệ này, như bảng dưới đây cho thấy, sử dụng các giá trị của năm 2012.

****

Hình . Tốc độ và chi phí theo từng công nghệ bộ nhớ

### 2.5.3. Level 1 cache

Trong kiến trúc đa bậc của bộ nhớ cache, các cấp cache được đánh số bắt đầu từ 1. Bộ nhớ cache cấp 1 (còn được gọi là L1 cache) là bộ nhớ cache đầu tiên mà processor tìm kiếm khi yêu cầu một lệnh hoặc mục dữ liệu từ bộ nhớ. Vì đây là điểm dừng đầu tiên trong quá trình tìm kiếm cache, nên L1 cache thường được xây dựng bằng công nghệ SRAM nhanh nhất có sẵn và được đặt càng gần mạch logic của processor càng tốt.

Việc tập trung vào tốc độ làm cho L1 cache đắt đỏ và tiêu tốn nhiều năng lượng, điều này có nghĩa là nó phải khá nhỏ so với bộ nhớ chính, đặc biệt là trong các ứng dụng có chi phí hạn chế. Ngay cả khi có kích thước khiêm tốn, một bộ nhớ cache cấp 1 có thể mang lại sự cải thiện đáng kể về hiệu suất so với một processor tương tự không sử dụng bộ nhớ cache.

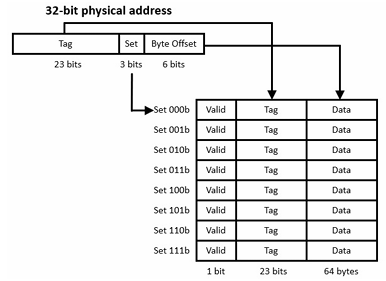
Bộ xử lý (hoặc MMU, nếu có) chuyển dữ liệu giữa DRAM và cache với các khối dữ liệu có kích thước cố định được gọi là dòng cache (cache lines). Các máy tính sử dụng module DDR DRAM thường có kích thước dòng cache là 64 bytes. Tất cả các cache ở nhiều bậc thường sử dụng chung một kích thước cache line.

Các bộ xử lý hiện đại thường chia bộ nhớ cache L1 thành hai phần có kích thước bằng nhau, một phần cho câu lệnh và một phần cho data. Cấu hình này được gọi là cache tách biệt (split cache). Trong split cache, bộ nhớ instruction cache cấp 1 được gọi là L1 I-cache và bộ nhớ data cache cấp 1 được gọi là L1 D-cache. Bộ xử lý sử dụng các bus riêng biệt để truy cập từng bộ nhớ cache, do đó triển khai một khía cạnh quan trọng của kiến trúc Harvard. Cách sắp xếp này tăng tốc độ thực thi lệnh bằng cách cho phép truy cập đồng thời cả câu lệnh và dữ liệu, giả sử rằng cả hai đều hit vào L1 cache.

### 2.5.4. Direct-mapped cache

Direct-mapped cache là một khối bộ nhớ được tổ chức dưới dạng mảng một chiều của các tập hợp cache, trong đó mỗi địa chỉ trong bộ nhớ chính ánh xạ tới một tập hợp duy nhất trong cache. Mỗi tập hợp cache bao gồm các mục sau:

* Một cache line, chứa một khối dữ liệu được đọc từ bộ nhớ chính.
* Một giá trị tag, chỉ vị trí trong bộ nhớ chính tương ứng với dữ liệu được lưu trong cache.
* Một valid bit, cho biết liệu tập hợp cache có chứa dữ liệu hay không.



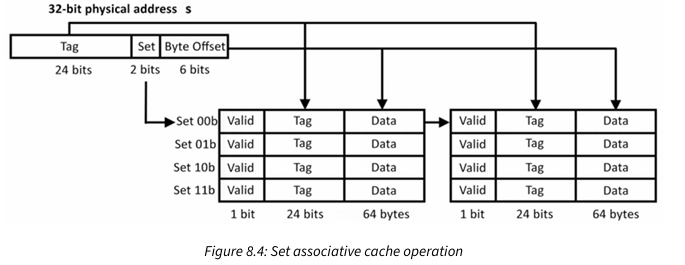
Hình . Mối quan hệ giữa một địa chỉ vật lý 32-bit và bộ nhớ cache

Hình ảnh này đại diện cho bộ nhớ cache lệnh, chỉ cho phép đọc. Bộ nhớ cache dữ liệu có thêm một bit nữa, được gọi là dirty bit. Nếu bit này được set, nó cho biết dữ liệu trong một dòng đã được sửa đổi và hệ thống phải ghi dữ liệu đó vào bộ nhớ trước khi cache line có thể được giải phóng để tái sử dụng.

Có những lúc bộ nhớ cache không chứa dữ liệu, chẳng hạn như ngay sau khi bật nguồn bộ xử lý. Valid bit cho mỗi tập hợp cache ban đầu được xóa để chỉ ra rằng không có dữ liệu trong tập hợp. Khi valid bit bị xóa, việc sử dụng tập hợp để tra cứu bị cản trở. Một khi dữ liệu đã được tải vào một cache set, phần cứng sẽ set valid bit.

### 2.5.5. Set associative cache

Trong bộ nhớ two-way set associative cache, bộ nhớ được chia thành hai bộ nhớ cache có kích thước bằng nhau. Mỗi bộ trong đó có một nửa số mục so với bộ nhớ direct-mapped cache có cùng tổng kích thước. Phần cứng sẽ kiểm tra cả hai bộ nhớ cache song song trong mỗi lần truy cập bộ nhớ, và một lần hit có thể xảy ra ở bất kỳ bộ nào. Sơ đồ sau minh họa việc so sánh đồng thời một tag địa chỉ 32-bit với các tag chứa trong hai L1 I-cache:

****

Hình . Sự vận hành của set associative cache

Khi xảy ra cache miss, memory management logic phải chọn bảng cache nào trong hai bảng sẽ được sử dụng làm điểm đến cho dữ liệu đang được tải từ DRAM. Một phương pháp phổ biến là theo dõi bảng liên quan nào trong hai bảng đã không được truy cập lâu nhất và ghi đè lên mục đó. Chính sách thay thế này, gọi là Least-Recently Used (LRU), yêu cầu hỗ trợ phần cứng để theo dõi cache line nào đã không được sử dụng lâu nhất. Chính sách LRU dựa trên nguyên tắc tính cục bộ theo thời gian, cho rằng dữ liệu không được truy cập trong một khoảng thời gian dài ít có khả năng được truy cập lại sớm.

Một phương pháp khác để chọn giữa hai bảng là chỉ đơn giản là xen kẽ giữa chúng trong các lần chèn cache liên tiếp. Phần cứng để thực hiện chính sách thay thế này đơn giản hơn so với chính sách LRU, nhưng có thể ảnh hưởng đến hiệu suất do việc chọn ngẫu nhiên dòng bị vô hiệu hóa. Việc chọn chính sách thay thế cache đại diện cho một sự đánh đổi giữa độ phức tạp của phần cứng và sự gia tăng hiệu suất.

Trong bộ nhớ two-way set associative cache, hai cache line từ các vị trí vật lý khác nhau có các trường Set tương ứng có thể tồn tại trong cache đồng thời, với điều kiện cả hai cache line đều hợp lệ. Đồng thời, so với bộ nhớ direct-mapped cache có cùng tổng kích thước, mỗi bộ two-way set associative cache chỉ bằng một nửa kích thước. Đây là một sự đánh đổi khác trong thiết kế: số lượng giá trị Set duy nhất có thể được lưu trữ trong bộ nhớ cache liên kết theo tập hợp 2 chiều bị giảm so với direct-mapped, nhưng nhiều cache line có trường Set giống nhau có thể được lưu trong cache đồng thời.

Cấu hình cache mang lại hiệu suất hệ thống tổng thể tốt nhất sẽ phụ thuộc vào các pattern truy cập bộ nhớ liên quan đến các loại hoạt động xử lý được thực hiện trên hệ thống.

Set associative cache có thể chứa nhiều hơn hai bộ nhớ cache như trong ví dụ này. Các bộ xử lý hiện đại thường có 4, 8 hoặc 16 bộ nhớ cache song song, được biết đến như 4-way, 8-way và 16-way set associative cache. Các bộ nhớ cache này được gọi là set associative vì một trường Tag địa chỉ liên kết với tất cả các cache line trong tập hợp cùng lúc.

Ưu điểm của multi-way set associative so với direct-mapped cache là nó thường có tỷ lệ cache hit cao hơn, dẫn đến hiệu suất hệ thống tốt hơn trong hầu hết các ứng dụng thực tế. Nếu multi-way set associative cache có hiệu suất tốt hơn one-way direct mapping, tại sao không tăng mức bậc associativity thậm chí nhiều hơn nữa? Nếu tăng đến giới hạn tối đa, quá trình này sẽ kết thúc với fully associative caching.

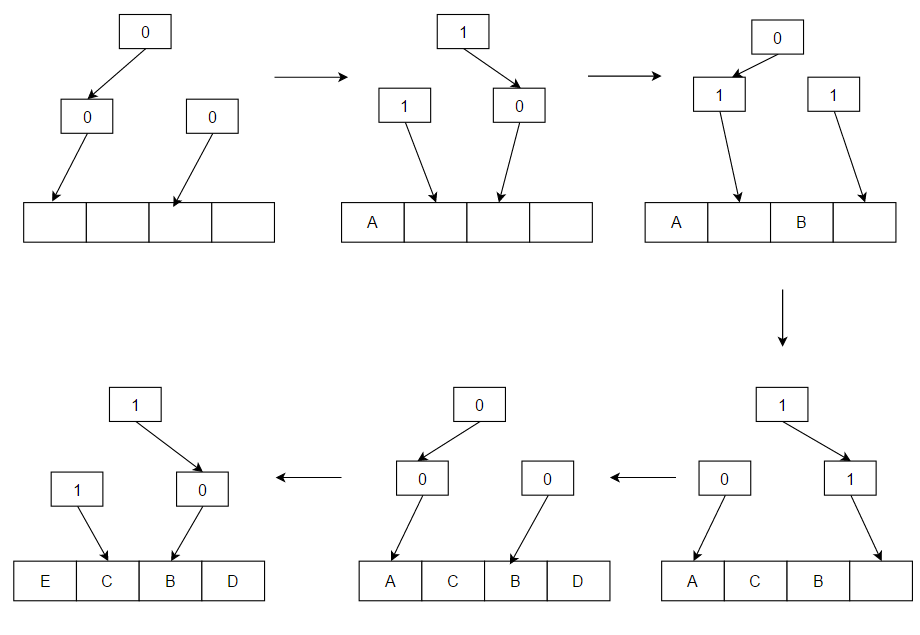
2.4.6. LRU cache

Các thuật toán thay thế cache được thiết kế một cache hiệu quả để thay thế dữ liệu trong cache khi không gian đã đầy. Least Recently Used (LRU) là một trong những thuật toán đó. Như tên gọi gợi ý, khi bộ nhớ cache đầy, LRU sẽ chọn dữ liệu được sử dụng ít nhất gần đây và loại bỏ nó để tạo chỗ cho dữ liệu mới. Độ ưu tiên của dữ liệu trong cache thay đổi theo nhu cầu của dữ liệu đó, tức là nếu một số dữ liệu được lấy hoặc cập nhật gần đây thì độ ưu tiên của dữ liệu đó sẽ thay đổi và được gán mức ưu tiên cao nhất, và độ ưu tiên của dữ liệu sẽ giảm đi nếu nó vẫn không được sử dụng qua các lần vận hành.

Pseudo-LRU hay PLRU là một hộ của các thuật toán cache cải thiện hiệu năng của thuật toán Least Recently Used (LRU) bằng cách thay thế các giá trị dựa trên các thước đo xấp xỉ về tuổi thay vì duy trì chính xác tuổi của từng giá trị trong cache. Tree-PLRU là một loại của LRU và có đạt được hiệu quả cao.

Tree-PLRU là một thuật toán hiệu quả để chọn một mục có khả năng không được truy cập gần đây nhất, dựa trên một tập hợp các mục và một chuỗi sự kiện truy cập đến các mục. Kỹ thuật này được sử dụng trong bộ nhớ cache CPU của Intel 486 và trong nhiều bộ xử lý thuộc dòng PowerPC, chẳng hạn như PowerPC G4 của Freescale được Apple Computer sử dụng.

Thuật toán hoạt động như sau: hãy xem xét một cây tìm kiếm nhị phân cho các mục đang xét. Mỗi nút của cây có một cờ một bit biểu thị đi sang trái để chèn một phần tử pseudo-LRU hoặc đi sang phải để chèn một phần tử pseudo-LRU. Để tìm một phần tử pseudo-LRU, duyệt qua cây theo các giá trị của cờ. Để cập nhật cây khi có một lần truy cập đến mục N, duyệt qua cây để tìm N và trong quá trình duyệt, đặt cờ nút để biểu thị hướng ngược lại với hướng đã đi.



Hình . Thuật toán tree-PLRU

Thuật toán này có thể không tối ưu vì nó chỉ là một phép xấp xỉ. Ví dụ, trong sơ đồ trên với các cache line A, C, B, D, nếu mẫu truy cập là: C, B, D, A, khi xảy ra việc loại bỏ, B sẽ được chọn thay vì C. Điều này xảy ra vì cả A và C đều nằm trong cùng một nửa, và việc truy cập A sẽ hướng thuật toán sang nửa còn lại, nơi không chứa cache line C.

### 2.5.6. Multilevel cache

Multilevel cache là một trong những kỹ thuật cải thiện hiệu suất của cache bằng cách giảm 'Miss Penalty'. Miss Penalty đề cập đến thời gian cần thiết để mang dữ liệu vào cache từ bộ nhớ chính khi xảy ra 'miss' trong cache.

Thời gian truy cập trung bình Multilevel cache (Tavg):

Trong đó:

* H1 là hit rate trong bộ nhớ cache L1.
* H2 là hit rate trong bộ nhớ cache L2.
* C1 là thời gian truy cập thông tin trong bộ nhớ cache L1.
* C2 là thời gian chuyển thông tin từ bộ nhớ cache L2 sang bộ nhớ cache L1.
* M là Miss penalty để chuyển thông tin từ bộ nhớ chính sang bộ nhớ cache L2.

Hệ thống multilevel cache trong các bộ xử lý hiện đại mang lại cải thiện hiệu suất đáng kể so với các hệ thống tương đương nhưng không có bộ nhớ cache. Bộ nhớ đệm cho phép các processor có tốc độ cao có thể chạy với việc truy cập bộ nhớ DRAM có độ trễ lớn. Mặc dù bộ nhớ cache làm tăng đáng kể độ phức tạp của thiết kế bộ xử lý và tiêu tốn nhiều diện tích die và năng lượng, các nhà cung cấp bộ xử lý đã xác định rằng việc sử dụng kiến trúc multilevel cach hoàn toàn xứng đáng với các chi phí này.

Ưu điểm của tổ chức multilevel cache:

* Giảm thời gian truy cập: Nhờ có nhiều cấp độ cache, thời gian truy cập dữ liệu được sử dụng thường xuyên giảm đáng kể. Dữ liệu đầu tiên sẽ được tìm trong bộ nhớ cache nhỏ nhất, nhanh nhất, và nếu không tìm thấy, sẽ tìm ở cấp độ cache lớn hơn và chậm hơn.
* Cải thiện hiệu suất hệ thống: Với thời gian truy cập nhanh hơn, hiệu suất hệ thống được cải thiện do CPU ít phải chờ dữ liệu được lấy từ bộ nhớ.
* Giảm chi phí: Nhờ có nhiều cấp độ cache, tổng lượng bộ nhớ cache cần thiết có thể được tối thiểu hóa. Điều này là do các cấp độ cache nhỏ hơn và nhanh hơn thường đắt hơn trên mỗi đơn vị bộ nhớ so với các cấp lớn hơn và chậm hơn.
* Tiết kiệm năng lượng: Vì dữ liệu được tìm kiếm trước tiên ở cấp độ cache nhỏ nhất, khả năng cao là dữ liệu sẽ được tìm thấy ở đó, giúp giảm tiêu thụ năng lượng của hệ thống.

Nhược điểm của tổ chức multilevel cache:

* Độ phức tạp: Việc thêm nhiều cấp độ cache làm tăng độ phức tạp của hệ thống cache và toàn bộ hệ thống. Điều này có thể làm cho việc thiết kế, gỡ lỗi và bảo trì hệ thống trở nên khó khăn hơn.
* Độ trễ cao hơn khi cache miss: Nếu dữ liệu không được tìm thấy trong bất kỳ cấp độ cache nào, nó sẽ phải được lấy từ bộ nhớ chính, điều này có độ trễ cao hơn. Độ trễ này có thể đặc biệt rõ ràng trong các hệ thống có cấu trúc cache nhiều cấp.
* Chi phí cao hơn: Mặc dù việc có nhiều cấp độ cache có thể giảm tổng lượng cache cần thiết, nó cũng có thể làm tăng chi phí hệ thống do cần thêm phần cứng cache.
* Vấn đề đồng bộ cache (cache coherence issues): Khi số lượng cấp độ cache tăng lên, việc duy trì tính nhất quán của dữ liệu giữa các cấp độ trở nên khó khăn hơn. Điều này có thể dẫn đến việc dữ liệu không nhất quán được đọc từ cache, gây ra các vấn đề trong hoạt động tổng thể của hệ thống.

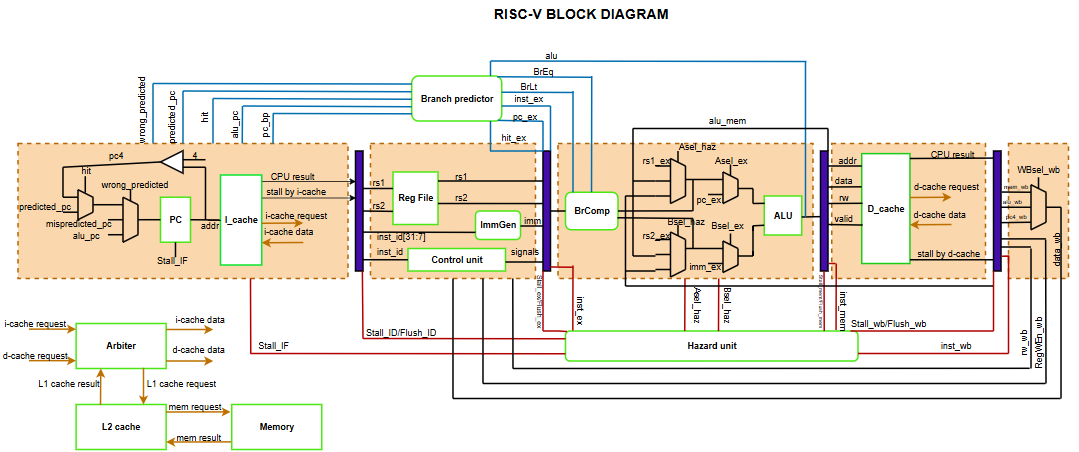
### 2.5.7. Level 2 cache

Các bộ xử lý hiện đại hiệu suất cao thường chứa một bộ nhớ cache L2 trên chip. Không giống như bộ nhớ cache L1, bộ nhớ cache L2 thường kết hợp cả instruction và data trong một vùng nhớ duy nhất, thể hiện kiến trúc von Neumann thay vì kiến trúc Harvard đặc trưng của cấu trúc split L1 cache.

Bộ nhớ cache L2 thường chậm hơn bộ nhớ cache L1, nhưng vẫn nhanh hơn nhiều so với việc truy cập trực tiếp vào DRAM. Mặc dù bộ nhớ cache L2 sử dụng các SRAM bit cell với cấu hình mạch cơ bản giống nhau, thiết kế mạch của L2 tập trung vào việc giảm diện tích mỗi bit trên die và mức tiêu thụ năng lượng so với thiết kế của L1. Những điều chỉnh này cho phép bộ nhớ cache L2 lớn hơn nhiều so với L1, nhưng đồng thời cũng khiến nó chậm hơn đáng kể.

# Thiết kế hệ thống

## 3.1. Thiết kế hệ thống tổng quan

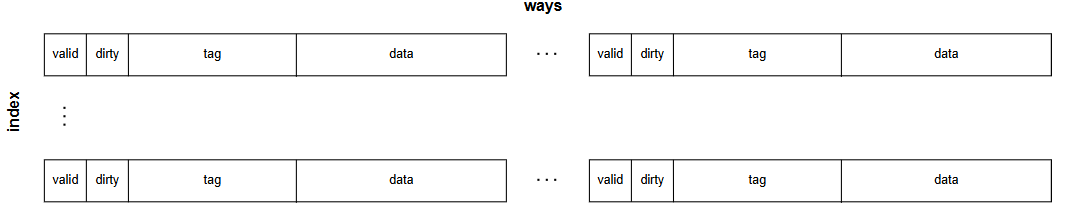


Hình . Sơ đồ khối toàn hệ thống

## 3.2. Thiết kế hệ thống chi tiết

### 3.2.1. Cấu trúc của Cache

Cache được thiết kế theo cấu trúc set-associative. Cấu trúc này được mô tả như hình dưới.

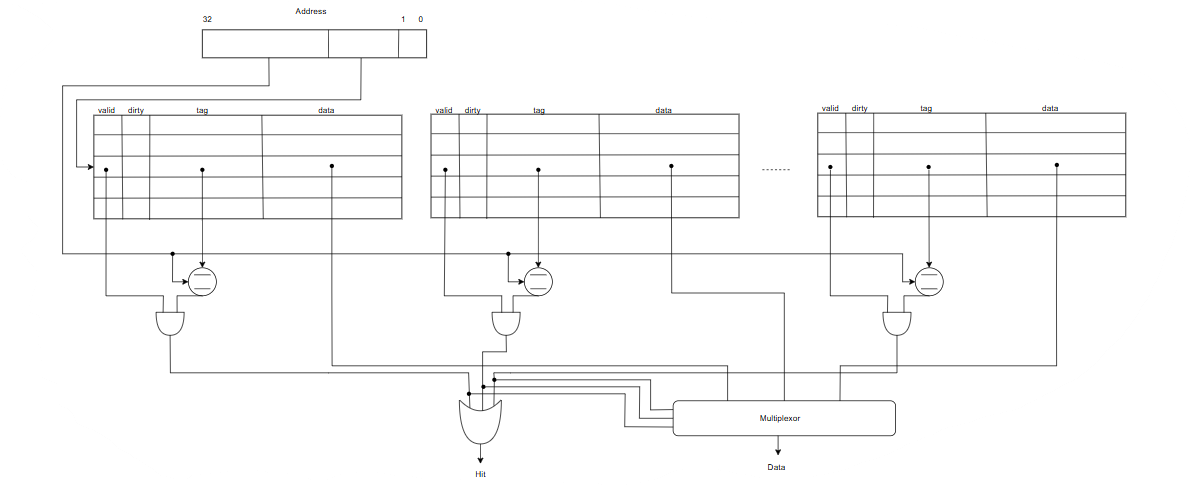


Hình . Cấu trúc của Cache

Cache được thiết kế theo nhiều set (xét theo chiều dọc), mỗi set chứa nhiều way (xét theo chiều ngang). Trong mỗi way có các bit sau:

* Valid: 1 bit. Đây là bit cho biết liệu block này có chứa dữ liệu hay không.
* Dirty: 1 bit. Đây là bit cho biết data trong block đã có trong cache ở tầng dưới hoặc memory hay chưa.
* Tag: Đây là các bit dùng để kiểm tra địa chỉ. Số bit của Tag được xác định bằng cách . Trong đó n là số bit offset của block, m là số bit index.
* Data: chứa dữ liệu được lưu trữ trong cache. Gồm 128 bits, tương đương 4 words, do đó n=2.
* Index: đây là các bit xác định địa chỉ của cache. Gồm m bits.

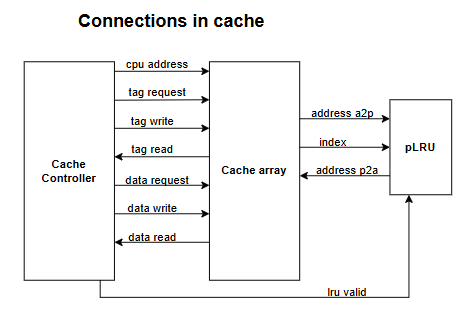
Dưới đây là cấu trúc mô tả cách truy cập cache qua các chỉ số của địa chỉ.



Hình . Cấu trúc truy cập Cache

### 3.2.1. Những kết nối trong Cache

Trong bộ nhớ Cache, có 3 submodule gồm cache controller, cache array, pLRU. Trong đó cache controller đóng vai trò điều khiển hoạt động của cache, giúp truyền địa chỉ và data từ CPU tới các ô nhớ trong cache và ngược lại. Cache array là khối chứa những dữ liệu của cache, bao gồm phần valid bit, dirty bit, tag, data. Khối pLRU đóng vai trò quyết định địa chỉ thay thế cho dữ liệu trong cache, với cache được thiết kế theo cấu trúc set-associative.



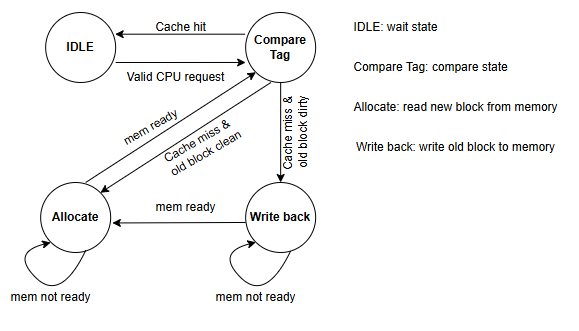
Hình . Những kết nối trong Cache

Các tín hiệu kết nối giữa những module được mô tả như sau:

* CPU address: 32 bit address, chứa địa chỉ từ CPU khi truy cập cache.
* Tag request, data request: chứa n bit index và 1 bit we, trong đó n là tổng số set trong cache, we là tín hiệu write enable cho phép tạo 1 tag/data mới trong cache.
* Tag write, tag read: gồm 1 bit valid, 1 bit dirty, n bit tag.
* Data write, data read: 128 bit (1 block, 4 words), chứa dữ liệu ghi/đọc từ cache.
* Address a2p, address p2a: n bit, trong đó n là số way của cache.
* Index: n bit, trong đó n là tổng số set trong cache.

### 3.2.1. Cache controller

Để điều khiển hoạt động của cache, phương pháp đơn giản nhất là sử dụng finite-state machine (FSM).

****

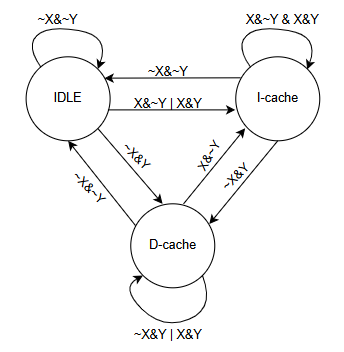
Hình . Cache controller FSM

Sơ đồ trên mô tả 4 trạng thái của cache controller:

* IDLE: Đây là trạng thái đợi cho tới khi tín hiệu yêu cầu ghi hoặc đọc từ bộ xử lý, trạng thái này sẽ chuyển đến trạng thái Compare Tag.
* Compare Tag: như tên gọi, trạng thái này kiểm tra xem yêu cầu đọc hoặc ghi là hit hay miss. Phần tag trong địa chỉ sẽ được dùng để so sánh với tag trong cache. Nếu dữ liệu trong khối cache được tham chiếu bởi phần chỉ số của địa chỉ là hợp lệ và phần tag của địa chỉ khớp với tag trong cache, thì đó là hit. Dữ liệu sẽ được đọc từ block đã chọn nếu đó là lênh load hoặc được ghi vào block đã chọn nếu đó là lệnh store. Sau đó, tín hiệu Cache Ready sẽ được set. Nếu đó là lệnh ghi, dirty bit sẽ được set lên 1. Lưu ý rằng một write hit cũng set valid bit và phần tag, mặc dù điều này có vẻ không cần thiết, nhưng nó được bao gồm vì tag là một bộ nhớ duy nhất, do đó để thay đổi dirty bit, chúng ta cũng cần thay đổi valid bit và tag. Nếu là hit và block là valid, FSM quay về trạng thái IDLE. Nếu miss, trước tiên cache tag được cập nhật và sau đó chuyển đến trạng thái Write-Back nếu block tại vị trí này có giá trị dirty bit là 1, hoặc chuyển đến trạng thái Allocate nếu giá trị là 0.
* Write-Back: trạng thái này ghi block 128-bit vào bộ nhớ bằng cách sử dụng địa chỉ được tạo từ tag và cache index. Trạng thái này sẽ giữ nguyên chờ tín hiệu Ready từ bộ nhớ. Khi ghi vào bộ nhớ hoàn tất, FSM sẽ chuyển sang trạng thái Allocate.
* Allocate: block mới sẽ được lấy (fetch) từ bộ nhớ. Trạng thái này sẽ giữ nguyên chờ tín hiệu Ready từ bộ nhớ. Khi đọc từ bộ nhớ hoàn tất, FSM sẽ chuyển sang trạng thái Compare Tag.

### 3.2.3. Cache arbiter

Cache arbiter là một bộ phân xử giúp quyết định chọn dữ liệu và địa chỉ từ icache hay dcache chuyển tiếp tới L2 cache. Dưới đây là sơ đồ khối thể hiện các trạng thái của khối Cache arbiter.



Hình . Sơ đồ khối Cache arbiter

Trong sơ đồ khối trên, các tín hiệu X và Y lần lượt là tín hiệu icache request valid và dcache request valid. Khối Cache arbiter sẽ ưu tiên instruction cache hơn khi cả 2 loại cache cùng truy cập đến L2 cache. Các trạng thái được mô tả như sau:

* IDLE: trạng thái khi không có L1 cache nào yêu cầu truy cập xuống L2 cache. Khi này giá trị L1 cache request sẽ bằng 0.
* I-cache: trạng thái cho phép instruction L1 cache truy cập vào L2 cache. Giá trị L1 cache request bằng i-cache request.
* D-cache: trạng thái cho phép data L1 cache truy cập vào L2 cache. Lúc này, giá trị L1 cache request bằng d-cache request.

# Mô phỏng và đánh giá thiết kế

# Kết luận và hướng phát triển

# Tài liệu tham khảo

"Computer Organization and Design RISC-V Edition: The Hardware Software Interface" by David A. Patterson and John L. Hennessy

https://riscv.org/