ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC & KĨ THUẬT MÁY TÍNH

KIẾN TRÚC MÁY TÍNH
BÀI TẬP CÁ NHÂN
-ĐỀ 3-

Giảng viên hướng dẫn: Nguyễn Xuân Minh

Sinh viên thực hiện: Đinh Lê Phúc Duy – 2210500

Thành phố Hồ Chí Minh, tháng 12, năm 2023

MỤC LỤC

I. CƠ SỞ LÍ THUYẾT VỀ BỘ NHỚ CACHE.

1. Bộ nhớ Cache là gì ?

- Cache là thành phần nhớ trong sơ đồ phân cấp bộ nhớ máy,hoạt động như thành phần trung gian giữa CPU và bộ nhớ chính.
- Cache có thể ở ngoài CPU (trong các hệ thống cũ) hoặc trong CPU (các hệ thống hiện đại).
- Cache thường có kích thước nhỏ: 16KB, 32KB,..., 128KB đối với các hệ thống cũ và 256KB, 512KB, 1MB,2MB,... với các hệ thống mới.
- Tốc độ của cache nhanh hơn so với tốc độ bộ nhớ chính còn chi phí tính trên bit của cache thì đắt hơn so với bộ nhớ chính.

2. Vai trò của bộ nhớ cache.

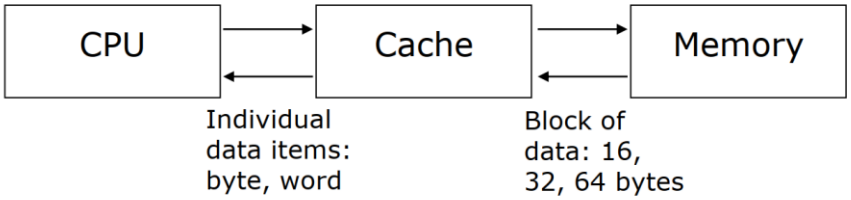
- Nâng cao hiệu năng hệ thống:
  - Cân bằng giữa tốc độ nhanh của CPU và tốc độ chậm của bộ nhớ chính (giảm số lượng truy cập trực tiếp của CPU vào bộ nhớ chính).
  - Thời gian truy cập trung bình của hệ thống bộ nhớ gần bằng thời gian truy cập cache.
- Giảm chi phí sản xuất:
  - Nếu 2 hệ thống có cùng hiệu năng thì hệ thống có cache sẽ rẻ hơn.
  - Nếu 2 hệ thống cùng chi phí, hệ thống có cache sẽ nhanh hơn.

3. Các nguyên lí cơ bản.

- Cục bộ về không gian:
  - Nếu một vị trí bộ nhớ được truy cập, thì khả năng các vị trí gần đó được truy cập trong thời gian gần tới là cao.
  - Áp dụng với các mục dữ liệu và các lệnh có thứ tự tuần tự theo chương trình.
  - Hầu hết các lệnh trong chương trình có thứ tự tuần tự, do đó cache đọc một khối dữ liệu trong bộ nhớ, mà bao gồm cả các phần tử xung quanh vị trí phần tử hiện tại được truy cập.
- Cục bộ về thời gian:
  - Nếu một vị trí bộ nhớ được truy cập, thì khả năng sẽ được truy cập trong thời gian tới là cao.
  - Áp dụng với các mục dữ liệu và các lệnh trong vòng lặp.
  - Cache đọc khối dữ liệu trong bộ nhớ bao gồm tất cả các thành phần trong vòng lặp.

4. Trao đổi dữ liệu.

- CPU đọc/ ghi từng mục dữ liệu riêng biệt từ/ vào cache.
- Cache đọc/ ghi khối dữ liệu từ/ vào bộ nhớ.



5. Tỉ lệ Hit và Miss của bộ nhớ Cache.

- Hit là sự kiện CPU truy cập tới mục dữ liệu mà tìm được trong cache.
  - Khả năng có thể xảy ra Hit được gọi là tỷ lệ hit H
  - 0<= H <= 1.
  - H càng cao thì cache càng tốt.
- Miss là sự kiện CPU truy cập tới mục dữ liệu mà không tìm thấy nó trong cache.
  - Khả năng xảy ra Miss gọi là tỷ lệ miss hay 1-H.
  - 0<= (1-H) <= 1.
  - Mong muốn tỷ lệ này thấp.

6. Tổ chức của bộ nhớ Cache.

- Ánh xạ trực tiếp (Direct Mapped)
  - Đơn giản, nhanh: khi biết địa chỉ bộ nhớ thì có thể truy cập nhanh chóng.
  - Khả năng ánh xạ cố định : xảy ra xung đột cao và tỉ lệ Hit thấp.
  - Cache: được chia thành n index (xem như là khối), được đánh dấu từ index 0 đến index n-1, mỗi index chứa m offset dùng để lưu địa chỉ của dữ liệu được dánh dấu từ offset 0 đến offset m-1.
  - Memory: Được chia thành nhiều Tag, mỗi Tag có kích thước bằng bộ nhớ Cache.
  - Ánh xạ: index i ở memory sẽ ánh xạ đến index i của cache, khi Tag đang được lưu ở cache khác Tag ở memory ánh xạ đến thì sẽ tiến hành đẩy dữ liệu ở cache ra và thay thế bằng dữ liệu mới.
  - Địa chỉ theo Direct Map:

Tag	Index	Offset

- Ánh xạ liên kết đầy đủ (Fully associative):
  - Phức tạp, chậm: phải tìm kiếm địa chỉ bộ nhớ và có thêm n bộ so sánh địa chỉ trong cache.
  - Khả năng ánh xạ linh hoạt : ít xung đột.
  - Cache: được chia thành nhiều khối nhưng lúc này ta không quan tâm đến index, lúc này offset chính là tổng số ô chứa địa chỉ của dữ liệu trong Tag.
  - Memory: Được chia thành nhiều Tag, mỗi Tag có kích thước như bộ nhớ Cache.
  - Ánh xạ: vì không quan tâm index nên sẽ ánh xạ Tag i của memory đến Cache, khi Tag đang được lưu ở cache khác Tag ở memory ánh xạ đến thì sẽ tiến hành đẩy dữ liệu ở cache ra và thay thế bằng dữ liệu mới.
  - Địa chỉ theo Fully Associative:

- Ánh xạ liên kết tập hợp/ theo bộ (k-ways set associative):
  - Phức tạp: cache được chia thành các way
  - Nhanh: ánh xạ trực tiếp được dùng có việc ánh xạ index,
  - Ít xung đột: ánh xạ từ Tag của memory đến way của cache là linh hoạt dẫn đến tỉ lệ Hit cao.
  - Cache: được chia thành k ways có kích thước bằng nhau. Mỗi way được chia thành n index (xem như là khối), được đánh dấu từ index 0 đến index n-1, mỗi index chứa m offset dùng để lưu địa chỉ của dữ liệu được đánh dấu từ offset 0 đến offset m-1, do đó số index sẽ giảm hơn k lần so với Direct Map.
  - Memory: Được chia thành nhiều Tag, mỗi Tag lúc này có kích thước bằng một way của cache.
  - Ánh xạ linh hoạt: Một Tag memory ánh xạ tới way bất kì của cache.
  - Ánh xạ cố định: index i ở memory của Tag j được ánh xạ đến index i ở way j của cache, index i của memory của Tag k sẽ được ánh xạ ở way k của cache. Khi các ways ở cache không còn chỗ trống để memory ánh xạ đến thì cache sẽ tiến hành đẩy dữ liệu của một way ra. Way được lựa chọn để đẩy ra có thể dùng theo các phương thức : FIFO (hàng đợi), LIFO (ngăn xếp), Random, LRU(Least Recently Use).
  - Địa chỉ theo set associative:

Ngoài ra trong bộ nhớ Cache sẽ có một bit đặc biệt gọi là bit V. Bit V trong bộ nhớ cache là bit hợp lệ, dùng để chỉ ra rằng phần tử cache đó có chứa dữ liệu hợp lệ hay không. Nếu bit V bằng 1, nghĩa là phần tử cache đó đã được cập nhật dữ liệu từ bộ nhớ chính và có thể sử dụng. Nếu bit V bằng 0 có nghĩa là phần tử cache đó chưa cập nhật hoặc đã bị xóa và không thể sử dụng. Bit V giúp tránh trường hợp đọc dữ liệu sai hoặc lỗi từ bộ nhớ cache. Và giá trị ban đầu của bit V là 0, tức là chưa có dữ liệu vào cache.

7. Hoạt động của bộ nhớ cache.

Khi có một HIT, dữ liệu được trả về cho bộ xử lý một cách nhanh chóng, vì nó đã được lưu trữ ở một vị trí gần bộ xử lý hơn.

Khi có một MISS, bộ xử lý phải đợi dữ liệu được tải từ bộ nhớ chính hoặc từ nguồn khác, và sau đó lưu vào bộ nhớ cache để sử dụng cho lần sau. Điều này làm chậm quá trình xử lý và giảm hiệu suất của hệ thống.

Như vậy khi truy xuất một dữ liệu từ cache, nếu cache đã chứa dữ liệu ấy thì tính hiệu trả về HIT, ngược lại nếu cache không tìm thấy dữ liệu thì sẽ tiến hành lưu những dữ liệu có cùng Tag và Index vào bộ nhớ cache.

II. ÁP DỤNG GIẢI BÀI TẬP.

Cho danh sách địa chỉ 32-bit truy xuất theo **địa chỉ word** như sau:
5, 172, 43, 37, 253, 88, 173, 5, 183, 44, 186, 252.

1. Nếu dùng bộ nhớ cache Direct-mapped có 32 block, mỗi block chứa **1 word**. Hãy xác định địa chỉ theo bit, từ đó suy ra các vùng tag, index lưu trữ vào cache. Cho biết trạng thái Hit/Miss của chuỗi truy xuất trên.

2. Làm lại câu (1) với bộ nhớ cache Direct-mapped có 16 block, mỗi block chứa **2 word**.

3. Hãy xác định tổng số bit bộ nhớ cần dùng để xây dựng bộ nhớ cache trong cả hai trường hợp. Biết rằng 1 phần tử cache sẽ chứa 1 bit V, các bit tag và dữ liệu.

Phân tích đề: Ở đây đề đã cung cấp cho ta một dãy các địa chỉ 32-bit truy xuất theo word, ta sẽ căn cứ vào cấu tạo của bộ nhớ Cache loại Direct-Mapped (Ánh xạ trực tiếp) được mô tả ở phần cơ sở lí thuyết để áp dụng vào giải quyết yêu cầu bài toán.

1. *Nếu dùng bộ nhớ cache Direct-mapped có 32 block, mỗi block chứa 1 word. Hãy xác định địa chỉ theo bit, từ đó suy ra các vùng tag, index lưu trữ vào cache. Cho biết trạng thái Hit/Miss của chuỗi truy xuất trên.*

Ở đây mỗi block chứa 1 word (tương ứng 2^0) nên số bit word offset là 0, mỗi Cache có 32 block (tương ứng 2^5) nên index được biểu diễn bằng 5 bits, các bit Tag sẽ được biểu diễn bởi 25 bits, và 2 bit còn lại là phần mở rộng byte offset của word. Phân phối các bit được mô tả rõ hơn sau đây:

Tag (25 bits)	Index (5 bits)	Mở rộng byte offset (2 bits)

Trước hết ta sẽ xác định địa chỉ theo bit của dãy đã cho và suy ra các vùng tag, index, ta sẽ được bảng sau:

	Tag	Index
5	00000000000000000000000000	00101
172	00000000000000000000000101	01100
43	00000000000000000000000001	01011
37	00000000000000000000000001	00101
253	00000000000000000000000111	11101
88	00000000000000000000000010	11000
173	00000000000000000000000101	01101
5	00000000000000000000000000	00101
183	00000000000000000000000001	10111
44	00000000000000000000000001	01100
186	00000000000000000000000101	11010
252	00000000000000000000000111	11100

Vì ánh xạ theo Direct-mapped nên mỗi lần ánh xạ sẽ thay thế cả một block nên các dữ liệu có cùng Tag và Index sẽ đi chung với nhau, từ đó ta có thể xác định được trạng thái Hit/Miss của chuỗi xuất hiện trên.

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Bit V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Khi truy xuất 5, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block phù hợp cùng với bit V như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						5										
Bit V	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Bit V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Khi truy xuất 172, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block phù hợp cùng với bit V như sau

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						5							172			
Bit V	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Bit V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Khi truy xuất 43, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block phù hợp cùng với bit V như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						5						43	172			
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Bit V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Khi truy xuất 37, Cache sẽ không tìm thấy trong lưu trữ nên trạng thái là MISS, nhưng ở vị trí ứng với index của 37 đã chứa dữ liệu khác nên cache tiến hành đẩy dữ liệu và cập nhật lại như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						37						43	172			
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Bit V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Khi truy xuất 253, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block phù hợp cùng với bit V như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						37						43	172			
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
														253		
Bit V	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Khi truy xuất 88, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block phù hợp cùng với bit V như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						37						43	172			
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
									88					253		
Bit V	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0

Khi truy xuất 173, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block phù hợp cùng với bit V như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						37						43	172	173		
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
									88					253		
Bit V	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0

Khi truy xuất 5, Cache sẽ không tìm thấy trong lưu trữ nên trạng thái là MISS, nhưng ở vị trí ứng với index của 5 đã chứa dữ liệu khác nên cache tiến hành đẩy dữ liệu và cập nhật lại như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						5						43	172	173		
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
									88					253		
Bit V	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0

Khi truy xuất 183, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block phù hợp cùng với bit V như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						5						43	172	173		
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
								183	88					253		
Bit V	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0

Khi truy xuất 44, Cache sẽ không tìm thấy trong lưu trữ nên trạng thái là MISS, nhưng ở vị trí ứng với index của 44 đã chứa dữ liệu khác nên cache tiến hành đẩy dữ liệu và cập nhật lại như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						5						43	44	173		
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
								183	88					253		
Bit V	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0

Khi truy xuất 186, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block phù hợp cùng với bit V như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						5						43	44	173		
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
								183	88		186			253		
Bit V	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0

Khi truy xuất 252, Cache sẽ không tìm thấy 252 trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block phù hợp cùng với bit V như sau:

Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						5						43	44	173		
Bit V	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0
Block	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
								183	88		186		252	253		
Bit V	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0

Qua quá trình trên, dễ thấy rằng các bit từ thứ 8 đến thứ 31 của các địa chỉ đều là bit 0 nên ta có thể biểu diễn theo dạng 3-5 (với 5 bit cuối là địa chỉ của index, 3 bit đầu cùng với 24 bit 0 còn lại là địa chỉ của Tag). Kí hiệu X(Y) có nghĩa là vị trí X trong cache thay đổi lần thứ Y.

	Địa chỉ (...3-5)	Tag	Index	Cache	HIT/MISS
5	...00000101	0	5	5(1)	MISS
172	...10101100	5	12	12(1)	MISS
43	...00101011	1	11	11(1)	MISS
37	...00100101	1	5	5(2)	MISS
253	...11111101	7	29	29(1)	MISS
88	...01011000	2	24	24(1)	MISS
173	...10101101	5	13	13(1)	MISS
5	...00000101	0	5	5(3)	MISS
183	...00110111	1	23	23(1)	MISS
44	...00101100	1	12	12(2)	MISS
186	...10110010	5	26	26(1)	MISS
252	...11111100	7	28	28(1)	MISS

$$-> Tỉ\ lệ\ MISS: 12/12.$$

2. *Làm lại câu (1) với bộ nhớ cache Direct-mapped có 16 block, mỗi block chứa 2 word.*

Ở đây mỗi block chứa 2 word (tương ứng 2^1) nên số bit word offset là 1, mỗi Cache có 16 block (tương ứng 2^4) nên index được biểu diễn bằng 4 bits, các bit Tag sẽ được biểu diễn bởi 25 bits, và 2 bit còn lại chính là phần mở rộng byte offset của word. Phân phối các bit được mô tả rõ hơn sau đây:

Tag (25 bits)	Index (4 bits)	Word offset (1 bit)	Mở rộng byte offset (2 bits)

Trước hết ta sẽ xác định địa chỉ theo bit của dãy đã cho và suy ra các vùng tag, index, ta sẽ được bảng sau:

	Tag	Index	Offset
5	00000000000000000000000000	0010	1
172	00000000000000000000000101	0110	0
43	00000000000000000000000001	0101	1
37	00000000000000000000000001	0010	1
253	00000000000000000000000111	1110	1
88	00000000000000000000000010	1100	0
173	00000000000000000000000101	0110	1
5	00000000000000000000000000	0010	1
183	00000000000000000000000001	1011	1
44	00000000000000000000000001	0110	0
186	00000000000000000000000101	1101	0
252	00000000000000000000000111	1110	0

Tương tự như ở câu a, ta có thể mô phỏng được bộ nhớ Cache như hình dưới, tuy nhiên mỗi lần truy xuất dữ liệu MISS, cache sẽ cập nhận dữ liệu có cùng tag và index với nhau vào cùng lúc.

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0																
	1																
Bit V		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Khi truy xuất 5, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp cùng với bit V như sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0																
	1			5													
Bit V		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Khi truy xuất 172, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp cùng với bit V như sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							172									
	1			5				173									
Bit V		0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0

Khi truy xuất 43, Cache sẽ không tìm thấy 43 trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp cùng với bit V như sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							172									
	1			5			43	173									
Bit V		0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0

Khi truy xuất 37, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp cùng với bit V như sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							172									
	1			37			43	173									
Bit V		0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0

Khi truy xuất 253, Cache sẽ không tìm thấy 253 trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp cùng với bit V như sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							172								252	
	1			37			43	173								253	
Bit V		0	0	1	0	0	1	1	0	0	0	0	0	0	0	1	0

Khi truy xuất 88, Cache sẽ không tìm thấy 88 trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp cùng với bit V như sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							172						88		252	
	1			37			43	173								253	
Bit V		0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	0

Khi truy xuất 173, Cache tìm thấy trong lưu trữ nên trạng thái là HIT.

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							172						88		252	
	1			37			43	173								253	
Bit V		0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	0

Khi truy xuất 5, Cache sẽ không tìm thấy trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp như sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							172						88		252	
	1			5			43	173								253	
Bit V		0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	0

Khi truy xuất 183, Cache sẽ không tìm thấy 183 trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp như sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							172						88		252	
	1			5			43	173					183			253	
Bit V		0	0	1	0	0	1	1	0	0	0	0	1	1	0	1	0

Khi truy xuất 44, Cache sẽ không tìm thấy 44 trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp như sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							44						88		252	
	1			5			43						183			253	
Bit V		0	0	1	0	0	1	1	0	0	0	0	1	1	0	1	0

Khi truy xuất 186, Cache sẽ không tìm thấy 186 trong lưu trữ, khi đó sẽ là trạng thái MISS và Cache sẽ được cập nhật lại với index ứng với số thứ tự block và offset phù hợp rnhư sau:

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							44						88	186	252	
	1			5			43						183			253	
Bit V		0	0	1	0	0	1	1	0	0	0	0	1	1	0	1	0

Khi truy xuất 252, Cache tìm thấy trong lưu trữ nên trạng thái là HIT.

Block		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset	0							44						88	186	252	
	1			5			43						183			253	
Bit V		0	0	1	0	0	1	1	0	0	0	0	1	1	0	1	0

Qua quá trình trên, dễ thấy rằng các bit từ thứ 8 đến thứ 31 của các địa chỉ đều là 0 nên ta có thể biểu diễn theo dạng 3-4-1 (với bit cuối là địa chỉ của offset, 4 bit kế cuối là địa chỉ của index, 3 bit đầu cùng với 24 bit 0 còn lại là địa chỉ của Tag). Kí hiệu X(Y) có nghĩa là vị trí X trong cache thay đổi lần thứ Y.

	Địa chỉ (...3-4-1)	Tag	Index	Offset	Cache	HIT/MISS
5	...00000101	0	2	1	2(1)	MISS
172	...10101100	5	6	0	6(1)	MISS
43	...00101011	1	5	1	5(1)	MISS
37	...00100101	1	2	1	2(2)	MISS
253	...11111101	7	14	1	14(1)	MISS
88	...01011000	2	12	0	12(1)	MISS
173	...10101101	5	6	1	6(1)	HIT
5	...00000101	0	2	1	2(3)	MISS
183	...00110111	1	11	1	11(1)	MISS
44	...00101100	1	6	0	6(2)	MISS
186	...10111010	5	13	0	13(1)	MISS
252	...11111100	7	14	0	14(1)	HIT

$$-> Tỉ\ lệ\ MISS: 10/12.$$

3. *Hãy xác định tổng số bit bộ nhớ cần dùng để xây dựng bộ nhớ cache trong cả 2 trường hợp. Biết rằng 1 phần tử cache sẽ chứa 1 bit V, các bit tag và dữ liệu.*

- Trường hợp bộ nhớ cache có 32 block, mỗi block chứa 1 word:

Tag (25 bits)	Index (5 bits)	Mở rộng byte offset (2 bits)

Khi này mỗi đơn vị cache sẽ bao gồm 1 bit V, các bit Tag (như bảng trên) và các bit dữ liệu(mỗi block chứa 1 word = 4 bytes tương ứng 32 bits), như vậy tổng số bit ở mỗi đơn vị cache là:

$$S = 1 + 25 + 32 = 58\ bits$$

Vậy tổng số bit của bộ nhớ cần dùng là:

$$S_{tong} = 32S = 1856\ bits.$$

- Trường hợp bộ nhớ cache có 16 block, mỗi block chứa 2 word:

Tag (25 bits)	Index (4 bits)	Word offset (1 bit)	Mở rộng byte offset (2 bits)

Khi này mỗi đơn vị cache sẽ bao gồm 1 bit V, các bit Tag (như bảng trên) và các bit dữ liệu(mỗi block chứa 2 word = 8 bytes tương ứng 64 bits), như vậy tổng số bit ở mỗi đơn vị cache là:

$$S = 1 + 25 + 64 = 90\ bits$$

Vậy tổng số bit của bộ nhớ cần dùng là:

$$S_{tong} = 16S = 1440\ bits.$$

Nhận xét: qua việc thực hiện bài toán trên, ta dễ dàng nhận thấy rằng việc tăng kích thước của mỗi Block đồng thời giảm đi số Block ở Cache sẽ giúp cải thiện được tỉ lệ HIT/MISS và tổng số bit của bộ nhớ cần dùng cũng giảm đi đáng kể, đổi lại ta sẽ tốn thêm thời gian cho việc so sánh khi truy xuất dữ liệu từ Cache.