



ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Báo cáo

Kiến trúc máy tính

Bài tập cá nhân

Họ và tên: Nguyễn Lê Hoàng Phúc

MSSV: 2212629

Lớp: L14

Email: phuc.nguyenlehoang707@hcmut.edu.vn

Tp. Hồ Chí Minh

08/12/2023

Đề: 4

Mô hình chung trong công tác xử lý địa chỉ trong cache:

Bước 1: Tìm block address:

$$Block\ address = \left\lfloor \frac{address_{main_memory}}{block\ size} \right\rfloor \quad (*)$$


Mục đích của bước này là khử đi các bit *offset* để thu được một *block address* chính xác.

Bước 2: Ánh xạ địa chỉ từ *Main memory* sang *Cache memory* bằng phương pháp modulo:

$$\begin{aligned} Cache_index &= (Block_address) \text{ modulo } (Number_of_blocks_in_cache) \\ &= (word_address_{main_memory}) \text{ modulo } (N) \end{aligned}$$

Nếu tại vị trí *cache_index* mà bộ nhớ trống (**miss**), thì nạp content (data) từ *main_memory* lên phần data của *cache_memory* tại *cache_index*, các giá trị *Tag*, *Valid* cũng sẽ được cập nhật. Ngược lại, so sánh *Tag* hiện tại với các bit cao của *block_address* để xác thực độ chính xác của dữ liệu, xem có đúng là dữ liệu chúng ta đang cần hay không; nếu có (**hit**), tiếp tục chương trình với đoạn dữ liệu đó, nếu không (**miss**), thực hiện các công việc tương tự như trường hợp block tại *cache_index* rỗng.

Câu a:

 Ta áp dụng mô hình trên để giải quyết vấn đề.

Với bước 1:

Điều kiện ban đầu của đề bài (bộ nhớ cache *Direct_mapped* có 32 blocks, mỗi block chứa 1 word) đã ám chỉ rằng vấn đề được giải quyết với *word_offset* thay vì *byte_offset*, có nghĩa ta chuyển hệ thống đơn vị cơ sở từ byte sang word. Theo đó sau khi

xác định được block, ta duyệt toàn bộ block với bước nhảy mỗi lần duyệt là 01 word (1 đơn vị) và do kích thước của mỗi block đúng bằng 01 word, nên nội dung trong 01 word của block đó chính là kết quả cần tìm và là duy nhất.

Dưới góc nhìn học thuật, ta có công thức:

$$B(offset) = \log_2(block_size) = \log_2(1) = 0 \quad (B(a): \text{hàm trả về số bit của } a)$$

Tức số bit cần thiết để biểu diễn *offset* là 0, hay không cần thiết biểu diễn *offset*. Như vậy, trong ngữ cảnh của bài toán đang xét, công thức (*) trở thành:

$$Block\ address = \left\lfloor \frac{word_address_{main_memory}}{words_per_block} \right\rfloor = \left\lfloor \frac{word_address_{main_memory}}{1} \right\rfloor = word_address_{main_memory}$$

Với bước 2: Áp dụng công thức để tính cache index với $N = 32$ (blocks).

$$Cache_index = (block_address) \bmod (N) = (word_address_{main_memory}) \bmod (32)$$

✚ Xác định số bit cho các vùng tag, index, offset:

$$B(offset) = 0(bits)$$

$$B(cache_index) = \log_2(N) = \log_2(32) = 5(bits) \quad (N: \text{so block})$$

$$B(tag) = 32 - [B(offset) + B(cache_index)] = 32 - (0 + 5) = 27(bits)$$

✚ Mô tả chi tiết quá trình xử lý địa chỉ word với bộ nhớ cache:

Quá trình 1: Xử lý địa chỉ word 5₁₀:

- Bước 1: <Đã giải thích>
- Bước 2: Chia lấy dư: $Cache_index = 5 \bmod 32 = 5$, tại block có chỉ số 00101:
 - + Nạp dữ liệu tại địa chỉ 5 trong main_memory vào vùng **Data** (do ban đầu cache rỗng);
 - + V = Yes (đánh dấu tính khả dụng của dữ liệu);
 - + Đánh dấu Miss (do dữ liệu chưa tồn tại trong cache).

STT	Hit / Miss	Cache index	V	Tag	Data
...	N
5	M	0 0101	Y	0000 0000 0000 0000 0000 0000 000- ----	Mem[5 ₁₀]
...	N

Quá trình 2: Xử lý địa chỉ word 164₁₀:

- Bước 1: <Đã giải thích>
- Bước 2: Chia lấy dư: $Cache_index = 164 \text{ modulo } 32 = 4$, tại block có chỉ số 00100:
 - + Nạp dữ liệu tại địa chỉ 4 trong main_memory vào vùng **Data** (do cache rỗng);
 - + V = Yes (đánh dấu tính khả dụng của dữ liệu);
 - + Đánh dấu Miss (do dữ liệu chưa tồn tại trong cache).

STT	Hit / Miss	Cache index	V	Tag	Data
...
4	M	0 0100	Y	0000 0000 0000 0000 0000 0000 101- ----	Mem[164 ₁₀]
...

Trong trường hợp tranh chấp xảy ra (2 dữ liệu địa chỉ từ bộ nhớ chính ánh xạ vào cùng một cache_index trong bộ nhớ đệm): Nếu 2 địa chỉ như nhau, tức nội dung của tại 2 địa chỉ đó không có sự khác biệt, thì tiếp tục chương trình với đoạn dữ liệu đó; Ngược lại, ghi đè nội dung tại địa chỉ mới lên nội dung tại địa chỉ cũ và tiếp tục chương trình với đoạn dữ liệu vừa được cập nhật (xem ví dụ ngay bên dưới).

Quá trình 4: Xử lý địa chỉ word 4₁₀:

- Bước 1: <Đã giải thích>
- Bước 2: Chia lấy dư: $Cache_index = 4 \text{ modulo } 32 = 4$, tại block có chỉ số 00100:

- + So sánh *Tag* trên block hiện tại với các bit cao tương ứng của địa chỉ đang xét và nhận thấy chúng khác nhau (00...0 101 ~ 00...0 000), do đó tiến hành ghi đè lên vùng data hiện tại;
- + Nạp dữ liệu tại địa chỉ 4 trong main_memory vào vùng **Data** (do tranh chấp xảy ra và 2 địa chỉ đang xét hoàn toàn khác nhau);
- + V = Yes (đánh dấu tính khả dụng của dữ liệu);
- + Đánh dấu Miss (do dữ liệu tồn tại trong cache mà không phải dữ liệu ta cần).

STT	Hit / Miss	Cache index	V	Tag	Data
...
4	M	0 0100	Y	0000 0000 0000 0000 0000 0000 000- ----	Mem[4 ₁₀]
...

Tiếp tục các bước như trên ta sẽ được kết quả mô phỏng bộ nhớ đệm cache ở trạng thái cuối cùng (sau khi thêm toàn bộ địa chỉ word theo yêu cầu) như [bảng 1](#).

Dưới đây là bảng tóm tắt kết quả câu a:

Địa chỉ word	Tag (27 bits)	Cache index (5 bits)	Hit/Miss
5	0000 0000 0000 0000 0000 0000 000	0 0101	Miss
164	0000 0000 0000 0000 0000 0000 101	0 0100	Miss
45	0000 0000 0000 0000 0000 0000 001	0 1101	Miss
4	0000 0000 0000 0000 0000 0000 000	0 0100	Miss
251	0000 0000 0000 0000 0000 0000 111	1 1011	Miss
90	0000 0000 0000 0000 0000 0000 010	1 1010	Miss
173	0000 0000 0000 0000 0000 0000 101	0 1101	Miss
164	0000 0000 0000 0000 0000 0000 101	0 0100	Miss
91	0000 0000 0000 0000 0000 0000 010	1 1011	Miss
44	0000 0000 0000 0000 0000 0000 001	0 1100	Miss
186	0000 0000 0000 0000 0000 0000 101	1 1010	Miss
252	0000 0000 0000 0000 0000 0000 111	1 1100	Miss

STT	Hit / Miss	Cache index	V ¹	Tag	Data
0		0 0000	N		
1		0 0001	N		
2		0 0010	N		
3		0 0011	N		
4	M	0 0100	Y	0000 0000 0000 0000 0000 0000 101- ----	Mem[164 ₁₀]
5	M	0 0101	Y	0000 0000 0000 0000 0000 0000 000- ----	Mem[5 ₁₀]
6		0 0110	N		
7		0 0111	N		
8		0 1000	N		
9		0 1001	N		
10		0 1010	N		
11		0 1011	N		
12	M	0 1100	Y	0000 0000 0000 0000 0000 0000 001- ----	Mem[44 ₁₀]
13	M	0 1101	Y	0000 0000 0000 0000 0000 0000 101- ----	Mem[173 ₁₀]
14		0 1110	N		
15		0 1111	N		
16		1 0000	N		
17		1 0001	N		
18		1 0010	N		
19		1 0011	N		
20		1 0100	N		
21		1 0101	N		
22		1 0110	N		
23		1 0111	N		
24		1 1000	N		
25		1 1001	N		
26	M	1 1010	Y	0000 0000 0000 0000 0000 0000 101- ----	Mem[186 ₁₀]
27	M	1 1011	Y	0000 0000 0000 0000 0000 0000 010- ----	Mem[91 ₁₀]
28	M	1 1100	Y	0000 0000 0000 0000 0000 0000 111- ----	Mem[252 ₁₀]
29		1 1101	N		
30		1 1110	N		
31		1 1111	N		

Bảng 1: Mô phỏng trạng thái bộ nhớ Cache trong quá trình hoạt động.²

¹ V: valid bit, N = No, Y = Yes.

² Trong thiết kế cấu trúc bộ nhớ Cache không tồn tại các cột *STT*, *Hit/Miss*, *Cache index*. Để phục vụ cho nhu cầu quan sát trực quan, chúng được thêm vào bảng và biểu diễn tương thích với từng dòng (từng block) của bộ nhớ Cache.

Câu b:

✚ Ta vẫn áp dụng mô hình ban đầu để giải quyết vấn đề.

Với bước 1:

Điều kiện của đề bài (bộ nhớ cache Direct_mapped có 16 blocks, mỗi block chứa 2 word) đã ám chỉ rằng vấn đề vẫn được giải quyết với *word_offset* thay vì *byte_offset* như câu a. Tuy nhiên sự khác biệt nằm ở việc: sau khi xác định được block, ta duyệt toàn bộ block với bước nhảy mỗi lần duyệt là 01 word (1 đơn vị) và do kích thước của mỗi block hiện tại là 02 word, nên sinh ra 2 trạng thái vị trí khác nhau trong cùng block. Đồng nghĩa với việc sau khi xác định được block, chúng ta cần xác định tiếp trạng thái vị trí nào phù hợp, và để phân biệt 2 trạng thái vị trí đó thì 1 bit trong địa chỉ word được sử dụng (0 cho trạng thái vị trí đầu tiên và 1 cho trạng thái vị trí còn lại).

Dưới góc nhìn học thuật, ta có công thức:

$$B(offset) = \log_2(block_size) = \log_2(2) = 1 \quad (B(a): \text{hàm trả về số bit của } a)$$

Tức số bit cần thiết để biểu diễn *offset* là 1, hay cần 1 bit để biểu diễn *offset*. Như vậy, trong ngữ cảnh của bài toán đang xét, công thức (*) trở thành:

$$Block_address = \left\lfloor \frac{word_address_{main_memory}}{words_per_block} \right\rfloor = \left\lfloor \frac{word_address_{main_memory}}{2} \right\rfloor$$

Với bước 2: Áp dụng công thức để tính cache index với $N = 32$ (blocks).

$$Cache_index = (block_address) \bmod (N) = \left(\left\lfloor \frac{word_address_{main_memory}}{2} \right\rfloor \right) \bmod (32)$$

✚ Xác định số bit cho các vùng tag, index, offset:

$$B(offset) = 1(bits)$$

$$B(cache_index) = \log_2(N) = \log_2(16) = 4(bits) \quad (N : \text{so block})$$

$$B(tag) = 32 - [B(offset) + B(cache_index)] = 32 - (1 + 4) = 27(bits)$$

🚦 Kết quả xử lý địa chỉ word với bộ nhớ cache:

Bảng tóm tắt kết quả câu b:

Địa chỉ word	Tag (27 bits)	Cache index (4 bits)	Offset (1 bit)	Hit/Miss
5	0000 0000 0000 0000 0000 0000 000	0 010	1	Miss
164	0000 0000 0000 0000 0000 0000 101	0 010	0	Miss
45	0000 0000 0000 0000 0000 0000 001	0 110	1	Miss
4	0000 0000 0000 0000 0000 0000 000	0 010	0	Miss
251	0000 0000 0000 0000 0000 0000 111	1 101	1	Miss
90	0000 0000 0000 0000 0000 0000 010	1 101	0	Miss
173	0000 0000 0000 0000 0000 0000 101	0 110	1	Miss
164	0000 0000 0000 0000 0000 0000 101	0 010	0	Miss
91	0000 0000 0000 0000 0000 0000 010	1 101	1	Hit
44	0000 0000 0000 0000 0000 0000 001	0 110	0	Miss
186	0000 0000 0000 0000 0000 0000 101	1 101	0	Miss
252	0000 0000 0000 0000 0000 0000 111	1 110	0	Miss

STT	Hit / Miss	Cache index	Offset	V ³	Tag	Data
0		0 000		N		
1		0 001		N		
2	M	0 010	0	Y	0000 0000 0000 0000 0000 0000 101- ----	Mem[164 ₁₀]
3		0 011		N		
4		0 100		N		
5		0 101		N		
6	M	0 110	0	Y	0000 0000 0000 0000 0000 0000 001- ----	Mem[44 ₁₀]
7		0 111		N		
8		1 000		N		
9		1 001		N		
10		1 010		Y		
11		1 011		N		
12		1 100		N		
13	M	1 101	0	Y	0000 0000 0000 0000 0000 0000 101- ----	Mem[186 ₁₀]
14	M	1 110	0	Y	0000 0000 0000 0000 0000 0000 111- ----	Mem[252 ₁₀]
15		1 111		N		

Bảng 2: Mô phỏng trạng thái bộ nhớ Cache trong quá trình hoạt động.⁴

³ V: valid bit. N = No, Y = Yes.

⁴ Trong thiết kế cấu trúc bộ nhớ Cache không tồn tại các cột *STT*, *Hit/Miss*, *Cache index*. Để phục vụ cho nhu cầu quan sát trực quan, chúng được thêm vào bảng và biểu diễn tương thích với từng dòng (từng block) của bộ nhớ Cache.

Câu c: Tổng số bit cần dùng để xây dựng bộ nhớ cache:

- Trong trường hợp câu a (32 blocks, mỗi block chứa 1 word):

$$\begin{aligned} Total_cache_bits &= [B(V) + B(tag) + B(data)] \times N \\ &= (1 + 27 + 32) \times 32 \\ &= 1920 \text{ (bits)} \end{aligned}$$

- Trong trường hợp câu b (16 blocks, mỗi block chứa 2 word):

$$\begin{aligned} Total_cache_bits &= [B(V) + B(tag) + B(data)] \times N \\ &= (1 + 27 + 32 \times 2) \times 16 \\ &= 1472 \text{ (bits)} \end{aligned}$$