8
9
1080 sp
نظري

كلية الهندسة المعلوماتية
السنة الثالثة

Cache Memory 2

د. خولة العلي

RB Informatics; 25/11/2024

بنيان الحواسيب 2

تحدثنا في المحاضرة السابقة عن الذواكر و عن ذاكرة الـ cache خاصة، حيث أنها تعمل كـ buffer لذاكرة الـ RAM.

و تحدثنا عن تنظيم ذاكرة cache، و عرفنا أن هناك ثلاثة أنواع، في محاضرتنا الحالية سنكمل شرح الأنواع و نتابع في الـ Cache.

↗ Cache Organization:

Associative Caches

الذاكرة المؤقتة التجميعية.

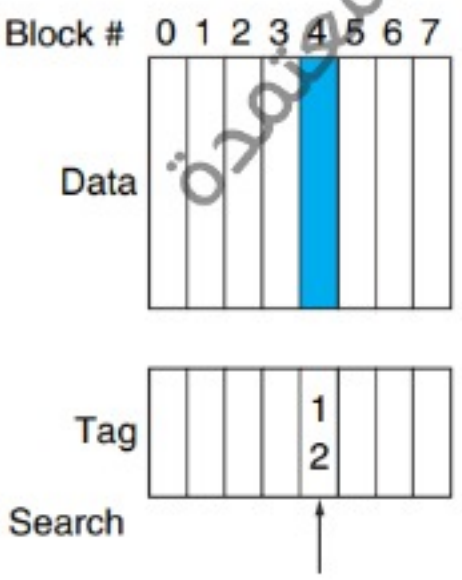
- في هذا النوع، يتم تقسيم الـ cache لعدة مجموعات كل مجموعة تحوي عدد من الأسطر، و عدد الأسطر في كل مجموعة يجب أن يكون نفسه.

A block can go in exactly one place in the cache 12 mod 8=4

- Each block in memory maps to unique set in cache.
- A block can be placed in any element of that set 12 mod 4=0

A block can go in any location in the cache.

Direct mapped

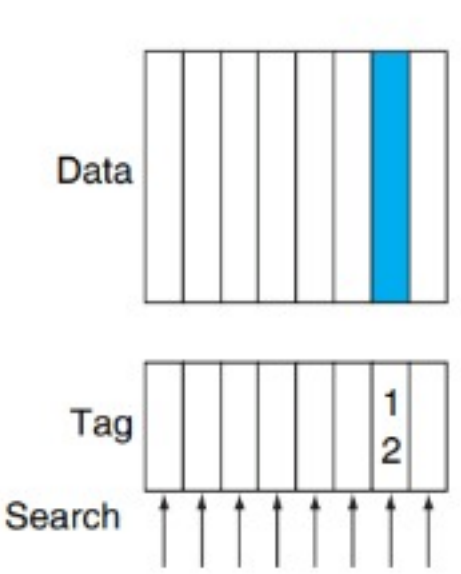


1-way
8 sets

Set associative

2-way
4 sets

Fully associative



8-way
1 sets

كما نرى في المخطط سابقاً، ذاكرة الـ cache فيها 8 أسطر، يمكن تقسيمها:

- 8 مجموعات في كل مجموعة سطر (بلوك) و هنا عدنا إلى Direct-mapped.
- مجموعتين في كل مجموعة أربعة أسطر .
- 4 مجموعات في كل مجموعة سطرين.
- مجموعة واحدة فيها 8 أسطر و هي حالة Fully Associative.

Fully Associative:

- و فيها تكون كل الأسطر ضمن مجموعة واحدة، أي أنه أي Block يأتي من ذاكرة الـ RAM يمكن أن يتجه إلى أي سطر في الـ Cache لا يوجد قيود.
- و في هذه الطريقة إذا أردنا أن نبحث عن شيء في الـ cache يجب أن نبحث عليها في كافة أسطر الـ cache، لأن كل الأسطر مجموعة واحدة.
- و نحتاج لعدد مقارنات مساوي لعدد أسطر الـ cache و ذلك بسبب إمكانية وجود المعلومة في أي سطر من الأسطر (و هي عملية مكلفة).

n-way set associative:

- كل مجموعة تحوي n سطراً.
- يتم تحديد رقم المجموعة، المطلوب في البحث بالقانون: باقي قسمة رقم الـ Block في الذاكرة على عدد مجموعات الـ cache.
- و يتم البحث في كل أسطر المجموعة المطلوبة بنفس الوقت.
- و نحتاج لعدد مقارنات مساوياً لعدد أسطر كل مجموعة (n) و هو أقل كلفة من الـ full associative.

Spectrum of Associativity

عدد التقسيمات الممكنة للـ associative cache

فرضاً لدينا ذاكرة cache بـ 8 أسطر، يمكننا تقسيمها كما يلي:

Each increase by a factor of 2 in associativity doubles the number of blocks per ser and halves the number of sets.

1 way ⟶ 8 sets

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

2 way ⟶ 4 sets

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

4 way ⟶ 2 sets

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

8 way ⟶ 1 sets

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Benefits of set associative:

1) تقلل معدل حالات الـ Miss عند البحث عن معلومة ما.
2) كل زيادة لعدد الأسطر في المجموعة الواحدة بنسبة عدد من مضاعفات الـ 2 هذا يؤدي لمضاعفة عدد الـ Blocks في المجموعة و ينقص عدد المجموعات للنصف.

Associativity example:

لدينا ذاكرة cache فيها عدد الأسطر 4

لنبحث عن العناوين التالية 8 ,6 ,0 ,8 ,0 في كل نوع من أنواع تنظيم الذاكرة التالية:

1) Direct mapped.
2) 2-way set associative.
3) Fully associative.

1. Direct mapped:

(Block number) modulo (Number of blocks in the cache)

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	Miss	Mem[0]			
8	0	Miss	Mem[8]			
0	0	Miss	Mem[0]			
6	2	Miss	Mem[0]		Mem[6]	
8	0	Miss	Mem[8]		Mem[6]	

أولاً: بما أن الـ cache هي 4 أسطر فإن عدد بتات الـ Index هو 2 لأن $4 = 2^2$ (التمثيل الثنائي).

ثانياً:

لمعرفة الموقع في الذاكرة فإننا ننظر لبتات الـ index.

فمثلاً الـ 8 يمثل ثنائياً $1000 \longleftarrow 00 = \text{index} \Longleftarrow$ في السطر (0).

الـ 6 تمثل 110 فهي في السطر (10) و هو السطر رقم (2)، (أو باستخدام قانون باقي القسمة).

و هذا ما نراه في الجدول أعلاه 😊

و بما لأنه أول مرور على البيانات فالحالة الابتدائية هي miss للكل.

2. 2-way set associative:

(Block number) modulo (Number of **sets** in the cache)

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	Miss	Mem[0]			
8	0	Miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	Miss	Mem[0]	Mem[6]		
8	0	Miss	Mem[8]	Mem[6]		

أولاً: هنا نرى مجموعتين و في كل مجموعة سطرين، نحصل على حقل index من باقي قسمة رقم الـ block المطلوب على عدد المجموعات، فمثلاً الـ block 8 باقي قسمته على 2 هو 0 فهو في المجموعة رقم (0) و لا يهم أي سطر من السطرين الموجودين في المجموعة.

ثانياً: نلاحظ هنا حالة hit و ذلك بسبب وجود الـ (0) ضمن المجموعة الواحدة مسبقاً و ضمن نفس السطر.

3. Fully associative:

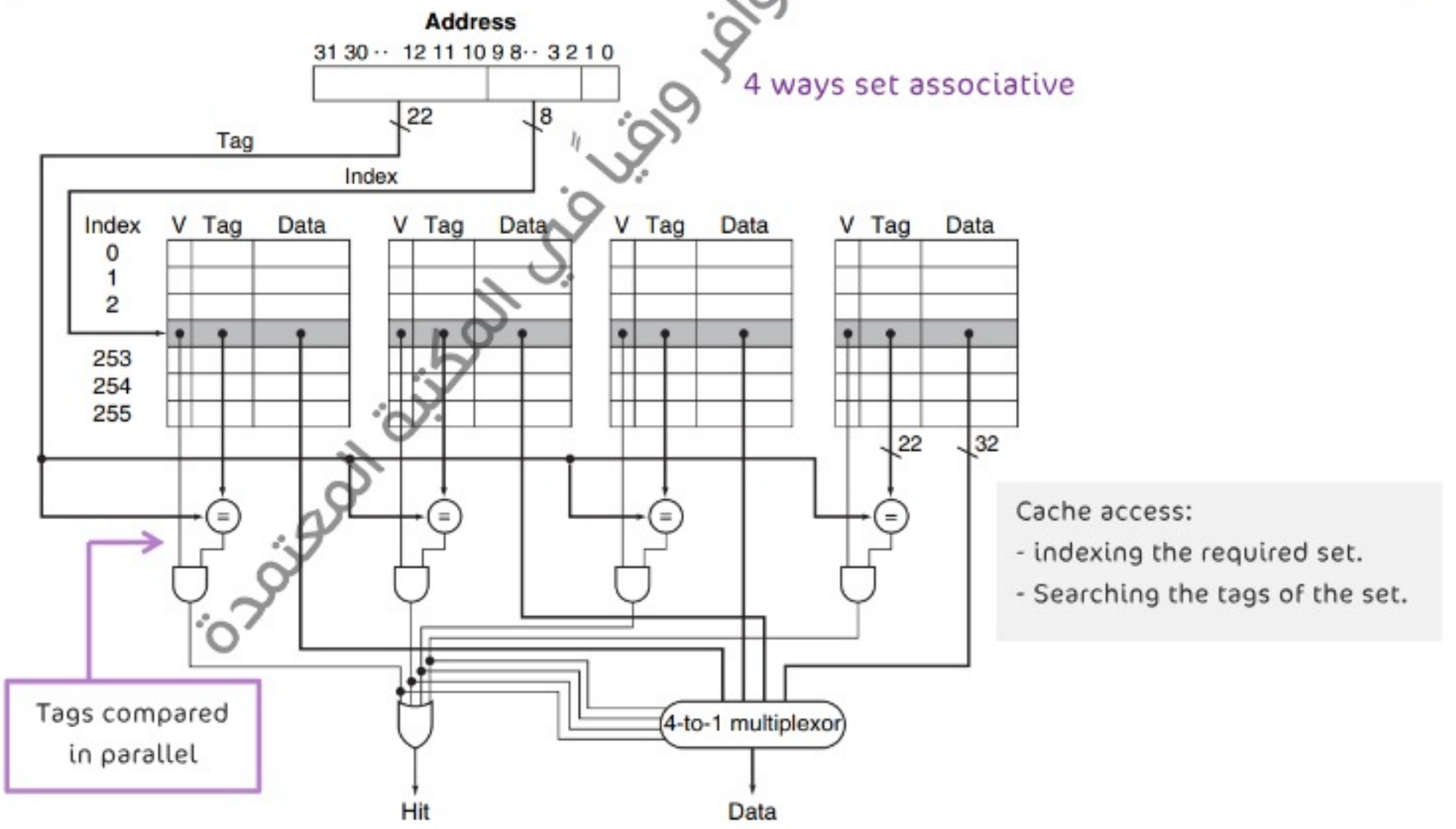
Block address		Hit/miss	Cache content after access			
0		Miss	Mem[0]			
8		Miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]		
6		Miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[6]	Mem[6]	

هنا نرى أنه يوجد حالتي Hit، و ذلك بسبب أنه يمكن للداتا أن توضع في أي سطر دون قيود، و هنا يتم البحث في كل الـ cache فإذا وُجد شيء مشابه (كما نرى في الجدول أعلاه) يعطي حالة hit.

في الطريقة الثانية: إذا لم يتم إيجاد مكان فارغ في المجموعة لوضع عنوان ما من الذاكرة فيتم استبدال عنوان ما حسب قاعدة الاستبدال Least recently used (LRU) (الشيء الأقل استخداماً مؤخراً).

فمثلاً (6) لم يكن لها مكان ننظر إلى المستخدم مؤخراً، 0 مستخدم مرتين و 8 تم استخدامها مرة فنستبدل مع (8).

Set Associative Cache Organization



و هنا يبين لنا عدد المقارنات التي تحتاجها في cache فيها مجموعات و لكل مجموعة 4 أسطر.

Handling Cache Misses

- عندما تحصل حالة Hit يعمل الـ cpu بتنفيذ التعليمات بشكل طبيعي.
- عندما تحصل حالة Miss:

1. يحصل حالة انتظار في توارد المعلومات.
2. يتم جلب الـ Block المطلوب منه المعلومات من المستوى الأعلى (ذاكرة RAM هنا) إلى ذاكرة cache.
3. نعيد جلب التعليمة من الـ cache بعد نسخها من الـ RAM.
4. يتم التنفيذ.

Four basic question on caches

1. أين يمكن وضع block في الـ cache؟

يتم وضعها حسب الأنواع التي درسناها سابقاً:

- Direct mapped.
- Associative.
- Fully associative.

2. كيف نجد block في الـ cache؟

إما حسب block address (tag-index) أو حسب block identification

3. أي block يتم استبداله في حالة miss؟

- Random : (يتم اختيار بلوك عشوائياً)
- FiFo: first in first out
- LRU: Least recently used (الأقل استخداماً حديثاً)
- NRU: newest recently used

4. ماذا يحدث أثناء الكتابة؟

عندما نعدل على الـ cache إما نعدل عليها فقط وعند حذف الشيء المعدل نأخذ النسخة المحدثة الى RAM أو يتم التعديل على cache وRAM في نفس الوقت.

Cache write

إذا كان المعالج يريد الكتابة على الذاكرة:

1) Write Hit (ذاكرة الـ cache تحوي الـ block المطلوب):

الذاكرة المؤقتة تحتوي على الـ block الذي نريد الكتابة عليه، ونحن نستطيع فقط التعديل على ال block الموجود في الـ cache، و لكن إذا تم التعديل ستصبح المعلومات في الذاكرة وال cache مختلفة، و هذا يسبب أخطاء (لأنه كما نعلم يتم نسخ المعلومات من الذاكرة RAM إلى cache و ليس نقلها).

بعض الطرق لمنع حدوث المشكلة:

Write Back
يتم التعديل على الـ cache، ثم يتم الانتظار إلى أن يتم نسخ الـ block المعدل إلى الذاكرة واستبداله، وللدلالة إلى أن الـ block تعدل هناك dirty bit إذا كانت قيمته (0) فهو لم يتم التعديل بعد أما إذا كان (1) فقد تم التعديل.

Write Buffer
يتم انتظار أن تكتب البيانات على الذاكرة، فيتابع المعالج عمله إلى أن تتم الكتابة و يتم عمل إيقافات لتتم هذه المهمة.

Write Through
يتم كتابة التعديل حقيقة في الـ cache، و هذا سيستغرق وقت أطول، لأن الذاكرة تحوي معلومة خاطئة.

2) Write Miss (الـ block المطلوب تعديله غير موجود في الـ cache):

الـ block المطلوب تعديله غير موجود في الـ cache.

No Write Allocate
توجيه طلب الكتابة لذاكرة RAM و لا يذهب لل cache، يتم تعديل الـ block المطلوب في الذاكرة دون وضعه في الـ cache.

Write Allocate
يتم تحميل الـ block المطلوب من الذاكرة إلى الـ cache ثم يتم التعديل عليه، و تحميل بت التعديل قيمة (1) لمعرفة أنه تم تعديل هذا الـ block، لا نعدله حاليًا في الذاكرة.
بل ننتظره حتى ينهي عمله و يتم استبداله.

Measuring cache performance

$$Memory\ Stall\ Cycles = Memory\ accesses\ per\ program \times miss\ rate \times miss\ penalty$$

الوقت الضائع المستغرق لجلب البيانات من الـ RAM و كتابتها في الـ cache.

- وهذا القانون يطبق فى عمليتى (الكتابة و القراءة).

ما الذي يحويه وقت التنفيذ في الـ cache؟

- دورات تنفيذ البرامج متضمنة وقت حالة الـ Hit.
- دورات التوقف في الذاكرة و بشكل رئيسي من حالات الـ miss في الـ cache.

Average Memory Access Time (AMAT)

و هو الوقت اللازم للوصول إلى الذاكرة بوجود Hits و misses

$$AMAT = Hit\ Time + (Miss\ rate * Miss\ Penalty)$$

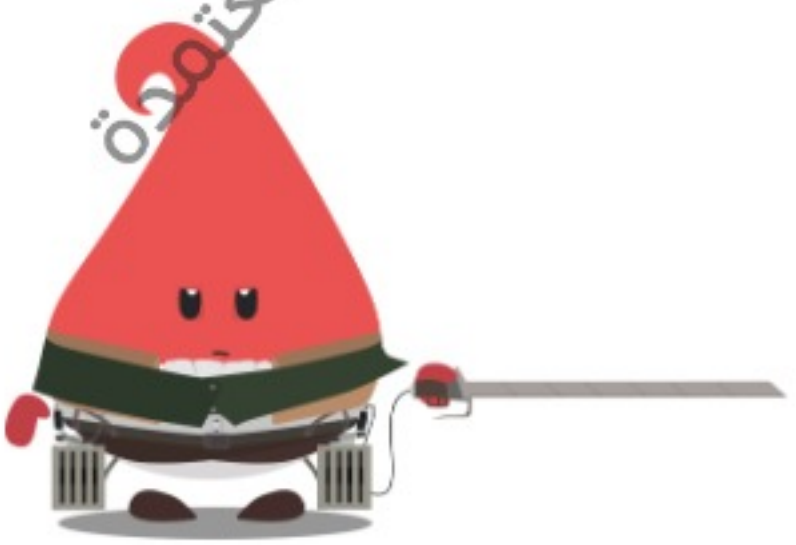
مثال:

أوجد AMAT من أجل المعطيات التالية:

$$cpu\ with\ (1ns) clock$$
$$Hit\ time = 1\ cycle$$
$$Miss\ penalty = 20\ cycle$$
$$miss\ rate = 5\%$$

$$\Rightarrow AMAT = 1 + (0.05 \times 20) = 2(ns)$$

Hit Time يزداد بزيادة حجم الـ cache و هو مهم لحساب الأداء.



Success occurs when your dreams get bigger than your excuses

Example problem:

احسب total bits المطلوب لتمثيل direct-mapped cache حيث: 128kb data و في كل block (سطر) يوجد 1-word و عنوان بسعة 32bits.

الحل:

$$cache\ data = 128kb = 128 \times 1024 = 2^7 \times 2^{10} = 2^{17}$$

- لكن كل block يحوي كلمة و الكلمة هنا حجمها $32bit$ أي $2^2 = 4\ byte$

$$\Rightarrow cache\ data = \frac{2^{17}}{2^2} = 2^{15}\ \ words, blocks$$

$$cache\ entry\ size = \text{عدد بتات البلوك} + tag\ \text{بتات} + 1\ valid\ bit$$

$$= 32(32 - 15 - 2) + 1 = 48\ bits$$

Index offset

$$\Rightarrow cache\ size = 2^{15} \times 48 = 2^{15} \times (2^5 \times 1.5) = 1.5 \times 2^{20} bits$$

$$= 1.5\ Mbits$$

2048

$$data\ bits\ in\ cache = 128kb \times 8 = 1Mbits$$ عدد بتات البيانات الموجودة

حولنا للمقارنة

$$\frac{total\ cache\ size}{actual\ cache\ data} = \frac{1.5}{1} = 1.5$$

The end