بسم الله الرحمن الرحيم

Computer Architecture

-6 & 5 –

ذاكرة الكاش

Cache Memory

2

تصميم الكاش Cache Design

الجدول التالي يوضح العناصر الأساسية لتصميم الكاش:

Cache Size	**Write Policy**
Mapping Function	Write through
Direct	Write back
Associative	Write once
Set associative	**Line Size**
Replacement Algorithm	**Number of Caches**
Least recently used (LRU)	Single or two level
First in first out (FIFO)	Unified or split
Least frequently used (LFU)	
Random	

1 - حجم الكاش Cache Size

كما ذكرنا من قبل فأنه يجب عند تحديد حجم الكاش الموازنة ما بين التكلفة و السرعة. حيث يجب أن يكون حجم الكاش صغيراً بما يكفي لجعل متوسط التكلفة للخانة الواحدة لنظام الذاكرة قريباً من تكلفة الخانة الواحدة للذاكرة الرئيسية (Main Memory) بمفردها، و في نفس الوقت يجب أن يكون حجم الكاش كبيراً بما يكفي لجعل متوسط السرعة لنظام الذاكرة قريباً من سرعة الكاش بمفرده. إضافة إلى ذلك توجد دوافع أخرى عديدة لتقليل حجم الكاش، على سبيل المثال كلما زاد حجم الكاش زاد بالضرورة عدد البوابات المنطقية المستخدمة في بنائه الأمر الذي يقلل من سرعته، و نتيجة لذلك تميل ذاكرة الكاش الأكبر حجماً إلى أن تكون أكثر بطئاً من نظيرتها الأصغر حجماً. كما أن محدودية المساحة المتاحة للكاش في شريحة المعالج تحد من حجم الكاش أيضاً، حيث نلاحظ أنه مع تطور تكنولوجيا بناء المعالجات و زيادة درجة التصغير أصبح في الإمكان وضع عدد أكبر من المكونات في نفس المساحة مما سمح بزيادة حجم الكاش.

2- Mapping Function

بما أن عدد خطوط الكاش (Cache Lines) أقل من عدد كتل الذاكرة (Memory Blocks) توجد حاجة لخوارزمية (Algorithm) لتحديد خط الكاش الذي ستوضع فيه كل كتلة من كتل الذاكرة. كما توجد حاجة أيضاً لطريقة لتحديد أي كتلة من كتل الذاكرة هي التي تشغل خطاً معيناً من خطوط الكاش حالياً. و الـ Mapping Functions المستخدمة هي: Direct، Associative، و Set Associative.

1- Direct Mapping

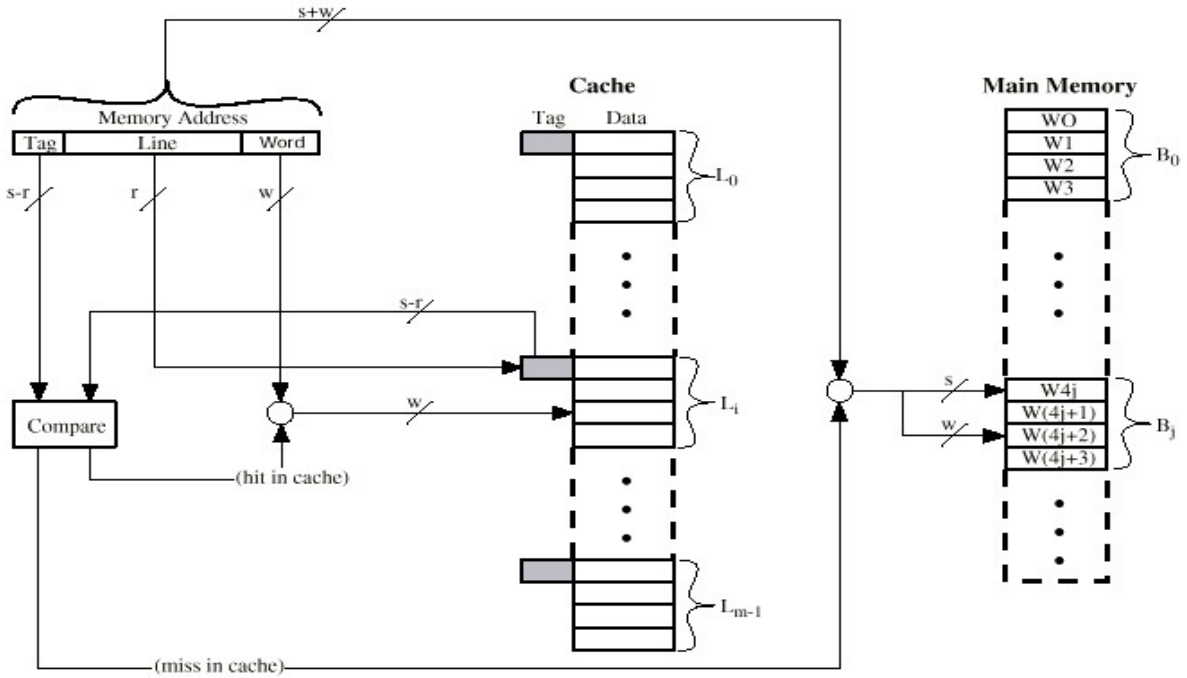
هو أبسط الأنواع، و فيه يتم تحديد خط معين من خطوط الكاش لكل كتلة من كتل الذاكرة. بمعنى أنه إذا كانت كتلة معينة من كتل الذاكرة موجودة بالكاش فإنها يجب أن تكون في خط الكاش المخصص لها. فإذا افترضنا أن عدد خطوط الكاش هو m و عدد كتل الذاكرة هو 2^s فإن توزيع كتل الذاكرة على خطوط الكاش يتم كالتالي:

Cache Line	Main Memory Blocks Assigned
0	$0, m, 2m, 3m, \dots, 2^s-m$
1	$1, m+1, 2m+1, 3m+1, \dots, 2^s-m+1$
2	$2, m+2, 2m+2, 3m+2, \dots, 2^s-m+2$
. . .	. . .
$m-1$	$m-1, 2m-1, 3m-1, 4m-1, \dots, 2^s-1$

للوصول إلى أي word موجودة بالكاش يتم تقسيم العنوان إلى ثلاثة حقول كما هو موضح بالشكل التالي:

tag	line	word
$s-r$	r	w

حيث تمثل الـ w خانة الدنيا من العنوان رقم الـ word في داخل الكتلة، و الـ r خانة التالية تمثل رقم خط الكاش المخصص للكتلة، أما الـ $s-r$ خانة الأخيرة من العنوان فتمثل الـ tag الذي يستخدم في التمييز بين كتل الذاكرة المخصص لها خط كاش معين و معرفة أي كتلة هي الموجودة حالياً في خط الكاش. الشكل التالي يوضح كيفية الوصول لـ word معينة باستخدام حقول العنوان الثلاثة:



يمتاز الـ Direct Mapping بالبساطة و انخفاض تكلفة الـ Hardware الخاص به. إلا أن عيبه الأساسي يكمن في تخصيص موقع ثابت بالكاش لكل كتلة ذاكرة. تصور أن برنامجاً ما يحتاج للرجوع لموقعي ذاكرة معينين يقعان ضمن كتلتي ذاكرة مختلفتين بصورة متكررة، و تصور أن كتلتي الذاكرة تم تخصيص نفس خط الكاش لهما. ماذا سيحدث؟

سيضطر البرنامج لاستبدال الكتلتين في الكاش باستمرار مما يقلل كثيراً من نسبة الإصابة (Hit Ratio)، و تسمى هذه الظاهرة بالـ Thrashing.

مثال:

وضح طريقة تقسيم العنوان لذاكرة رئيسية سعتها 16 MByte و حجم الكتلة بها هو 4 Byte، و ذلك إذا كان حجم الكاش 64 KByte و الـ Mapping Function المستخدم هو Direct Mapping.

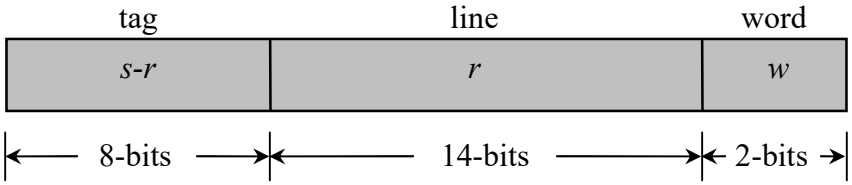
الحل:

عدد كتل الذاكرة = سعة الذاكرة/حجم الكتلة = $\frac{16\ \text{MByte}}{4\ \text{Byte}} = 4\ \text{M} = 2^{22} = 2^2\ 2^{20} \Leftarrow s = 22$

عدد خطوط الكاش = سعة الكاش/حجم الكتلة = $\frac{64\ \text{KByte}}{4\ \text{Byte}} = 16\ \text{K} = 2^{14} = 2^4\ 2^{10} \Leftarrow r = 14$

حجم الكتلة = $4 = 2^2 \Leftarrow w = 2$

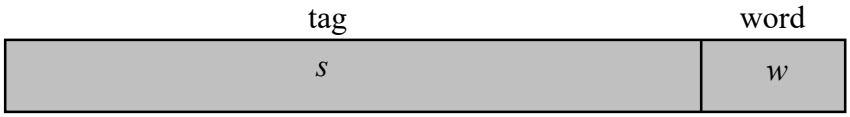
و عليه يتم تقسيم العنوان كالتالي:



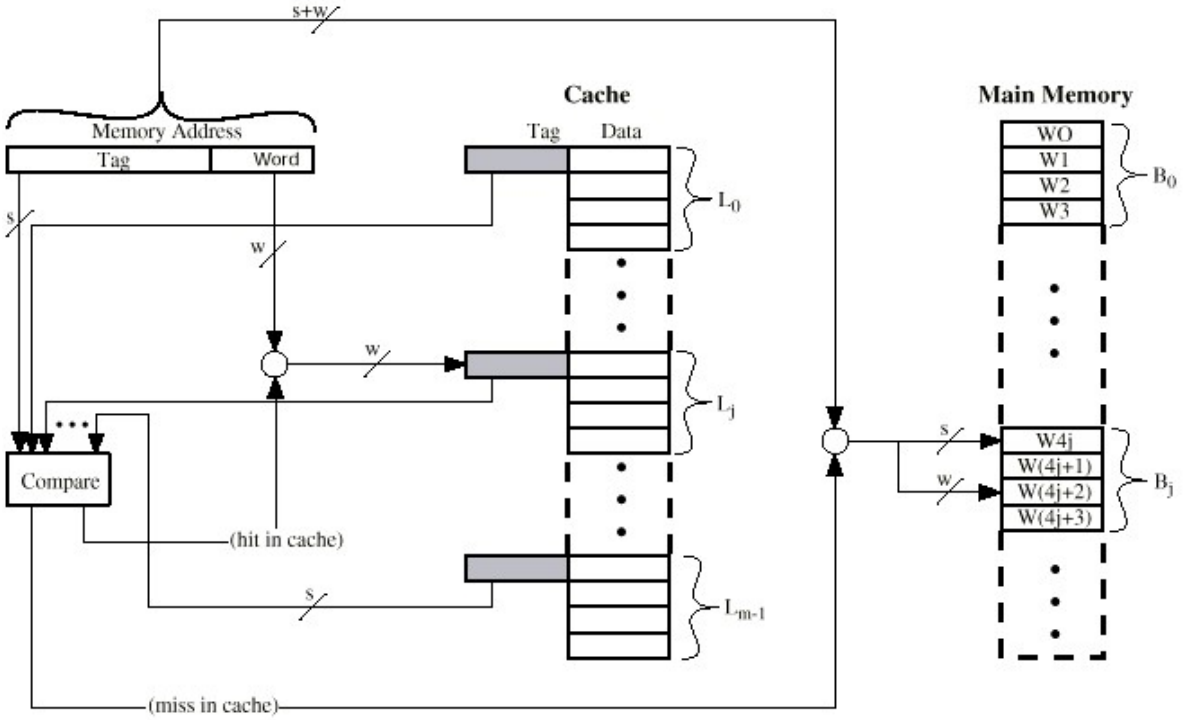
Associative Mapping –2

يتغلب هذا النوع من الـ Mapping على ظاهرة الـ Thrashing التي لاحظناها في الـ Direct Mapping، و ذلك بعدم تخصيص خط معين بالكاش لكتلة الذاكرة و إنما السماح بوضع كتلة الذاكرة في أي خط من خطوط الكاش.

للوصول إلى أي word موجودة بالكاش يتم تقسيم العنوان إلى حقلين فقط كما هو موضح بالشكل التالي:



حيث تمثل الـ w خانة الدنيا من العنوان رقم الـ word في داخل الكتلة، و تمثل الـ s خانة المتبقية الـ tag الذي يستخدم في التمييز بين كتل الذاكرة معرفة أي كتلة هي الموجودة حالياً في خط الكاش. الشكل التالي يوضح كيفية الوصول لـ word معينة باستخدام حقلي العنوان:



لاحظ أن عدد خانات الـ tag هنا أكبر من عدد خاناته في الـ Direct Mapping، كما أنه في حين كان يتم مقارنة الـ tag لخط واحد فقط من خطوط الكاش في حالة الـ Direct Mapping يجب هنا مقارنة الـ tag لجميع خطوط الكاش (على التوازي) الأمر الذي يرفع كثيراً من درجة تعقيد و تكلفة الـ Hardware لهذا النوع من الـ Mapping.

3- Set Associative Mapping

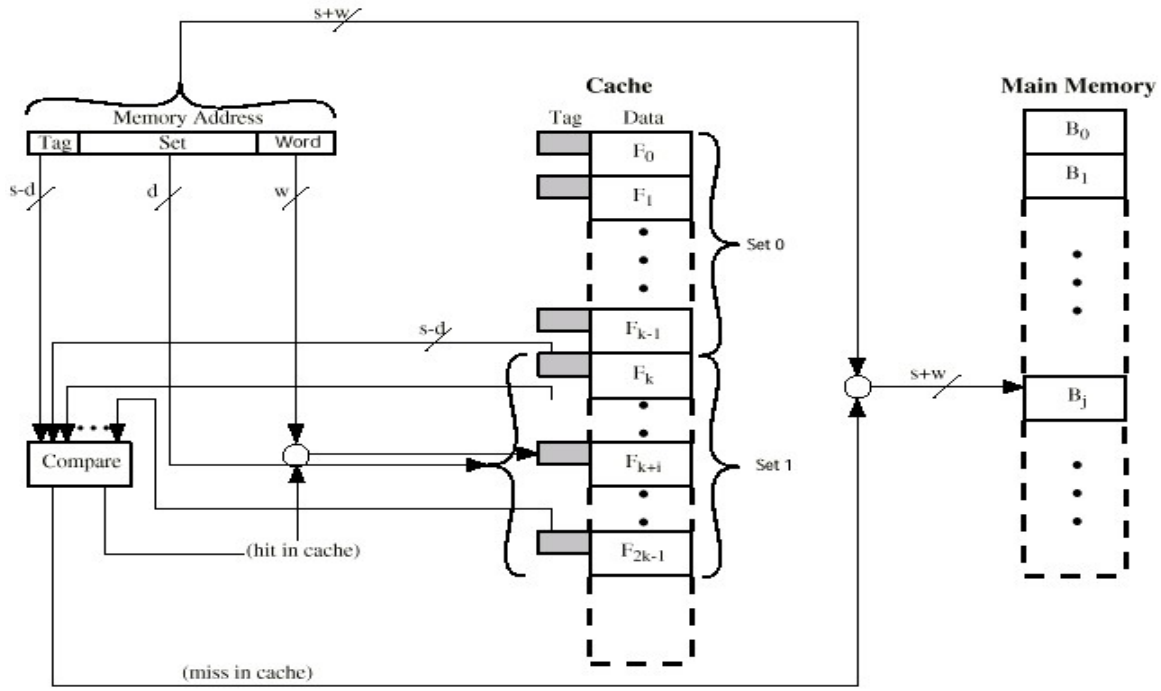
هذا النوع من الـ Mapping هو حل وسط يحاول الإستفادة من ميزات النوعين السابقين مع التقليل من عيوبهما. في هذا النوع يتم تقسيم الكاش إلى عدد من المجموعات (Sets)، كل مجموعة منها مكونة من k خطاً. و يتم تحديد مجموعة لكل كتلة من كتل الذاكرة، و لكن يمكن وضع الكتلة في أي خط من الخطوط المكونة للمجموعة.

عدد خطوط المجموعة الأكثر استخداماً هو $k = 2$، و يطلق على الـ Mapping هنا تسمية 2-Way Set Associative Mapping. و يؤدي إلى تحسن كبير في نسبة الإصابة (Hit Ratio) مقارنة بالـ Direct Mapping. كما يستخدم أيضاً العدد $k = 4$ و يطلق عليه 4-Way Set Associative Mapping، و يؤدي إلى تحسن طفيف في نسبة الإصابة مقارن بالـ 2-Way Set Associative Mapping.

للوصول إلى أي word موجودة بالكاش في هذا النوع من الـ Mapping يتم تقسيم العنوان إلى ثلاثة حقول كما هو موضح بالشكل التالي:

tag	set	word
s-d	*d*	*w*

حيث تمثل الـ w خانة الدنيا من العنوان رقم الـ word في داخل الكتلة، و الـ d خانة التالية تمثل رقم المجموعة المخصصة للكتلة، أما الـ s-d خانة الأخيرة من العنوان فتمثل الـ tag الذي يستخدم في التمييز بين كتل الذاكرة المخصص لها مجموعة معينة و معرفة أي كتلة هي الموجودة حالياً في المجموعة و مكان وجودها داخلها. الشكل التالي يوضح كيفية الوصول لـ word معينة باستخدام حقول العنوان الثلاثة:



تمرين 1:

قارن ما بين عدد خانات الـ tag و عملية المقارنة لـ tags في أنواع الـ Mapping الثلاثة.

تمرين 2:

يمكن اعتبار أن كل من الـ Direct Mapping و الـ Associative Mapping هي حالات خاصة من الـ Set Associative Mapping. وضح ذلك.

3- خوارزمية الإستبدال Replacement Algorithm

عند نقل كتلة من كتل الذاكرة إلى الكاش فإنها يجب أن تحل محل إحدى الكتل الموجودة به. السؤال الآن هو أي من الكتل الموجودة بالكاش سيتم استبدالها؟

بالنسبة لـ Direct Mapping لا يوجد خيار حيث أن الكتلة المنقولة بالكاش لها خط كاش محدد مسبقاً و سيتم بالضرورة استبدال الكتلة الموجودة بذلك الخط. أما بالنسبة لـ Associative Mapping و الـ Set Associative Mapping فيوجد خيار و لابد من خوارزمية للإستبدال. و توجد عدة خوارزميات للإستبدال أشهرها:

- Least Recently Used (LRU)
- First-In-First-Out (FIFO)
- Least Frequently Used (LFU)
- Random

4- Line Size

يعتبر حجم الخط (Line Size) من الخصائص المهمة لتصميم الكاش، حيث أنه عندما يتم وضع كتلة معينة من كتل الذاكرة في الكاش لا يتم، كما نعلم، إحضار الـ word المطلوبة فقط من الذاكرة و إنما يتم أيضاً إحضار مجموعة من الـ words المجاورة لها. فبزيادة حجم الكتلة تبدأ نسبة الإصابة (Hit Ratio) في البداية بالزيادة بسبب ظاهرة تموضع المراجع (Locality of Reference)، التي تنص على أن المواقع المجاورة لـ word التي تم الرجوع إليها غالباً ما يتم الرجوع إليها أيضاً في المستقبل القريب. فكلما زاد حجم الكتلة تم إحضار المزيد من البيانات المفيدة من الذاكرة إلى الكاش. و لكن مع الإستمرار في زيادة حجم الكتلة أكثر من ذلك تبدأ نسبة الإصابة (Hit Ratio) بالتراجع، و يصبح احتمال الحاجة لاستخدام البيانات الجديدة أقل من احتمال الحاجة لإعادة استخدام البيانات التي تم استبدالها. و يعود ذلك للسببين التاليين:

1. كلما زاد حجم الكتلة قل عدد الكتل التي يمكن وضعها بالكاش. و بما أن أي كتلة جديدة يتم إحضارها تحل محل إحدى الكتل الموجودة فعلاً بالكاش فإن العدد القليل للكتل يؤدي إلى أن يتم استبدال الكتل بعد فترة قصيرة من إحضارها، قبل إنتفاء الحاجة لها.
2. كلما زاد حجم الكتلة زاد بعد الـ words الإضافية الموجودة بها عن الـ word المطلوبة مما يقلل من احتمال الحاجة للرجوع لهذه الـ words الإضافية.

بناء على ما سبق فإن العلاقة ما بين حجم الكتلة و نسبة الإصابة (Hit Ratio) هي علاقة معقدة، و تعتمد على خصائص التموضع (Locality) للبرنامج الذي يتم تنفيذه. و لم يتم التوصل لقيمة مثلى لحجم الكتلة و لكن وضح بالتجربة أن حجم كتلة يتراوح ما بين 8 و 32 يعتبر مناسباً و ينتج عنه مستويات أداء قريبة من المثلى.

5- Number of Caches

عند بداية ظهور ذاكرة الكاش عادة ما كان يوجد مستوى واحد من الكاش، و لكن حديثاً أصبح من الشائع استخدام عدة مستويات من الكاش، و بعض هذه المستويات قد يكون داخل شريحة المعالج نفسها و بعضها الآخر قد يكون خارج المعالج. و بالتالي فإنه يجب عند تصميم الكاش تحديد عدد مستويات الكاش (Number of Levels) ، و ما إذا كانت تلك المستويات داخلية أو خارجية، و تحديد ما إذا كان الكاش موحداً أو مقسوماً (Unified or Split).

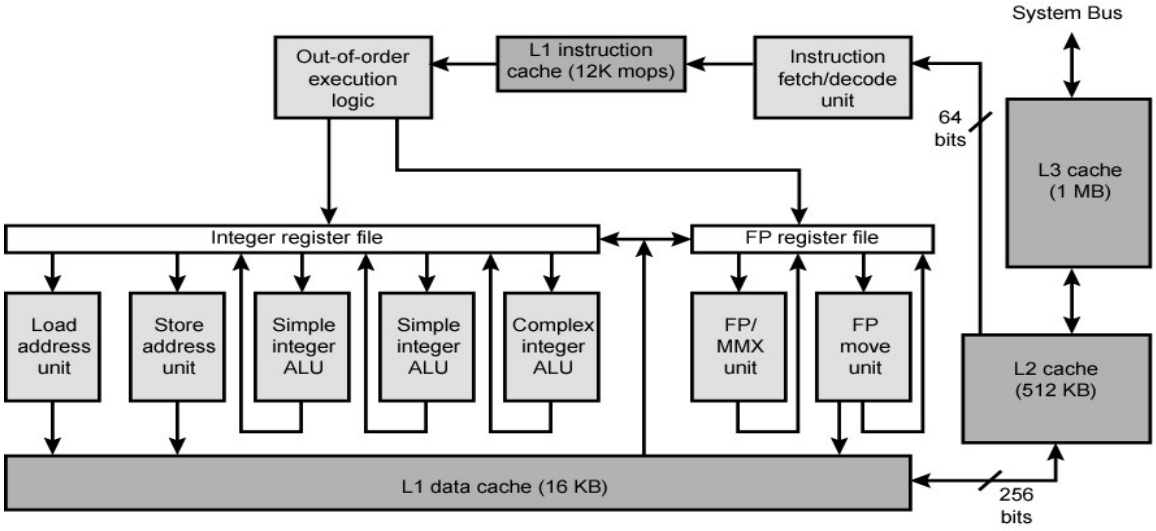
عند بداية ظهور الكاش كان يوجد خارج شريحة المعالج و كان الوصول إليه يتم عبر ناقل النظام (System Bus). و لكن مع تطور تكنولوجيا المعالجات أصبح في الإمكان وضع الكاش داخل شريحة المعالج نفسها. و الكاش الداخلي (On-Chip Cache) يزيد من سرعة الدخول بسبب قصر المسافات مقارنة بالكاش الخارجي، كما يقلل كثيراً من حاجة المعالج لاستخدام ناقل النظام (System Bus) و يزيد بالتالي من سرعة المعالجة و يحسن من مستوى الأداء.

بعد أن تم نقل الكاش إلى داخل شريحة المعالج ظهرت مشكلة محدودية حجم هذا الكاش الداخلي، بسبب محدودية المساحة المتاحة له داخل شريحة المعالج، و أصبحت هناك حاجة لمستوى آخر من الكاش أكبر حجماً. و عليه أطلق على الكاش الداخلي تسمية L1 Cache و على الكاش الخارجي الجديد الأكبر حجماً تسمية L2 Cache. و حيث أن الدخول على الـ L2 Cache كان يتم عبر ناقل النظام (System Bus) ظهرت مشكلة زيادة الحاجة لاستخدام الناقل مرة أخرى و أصبح ناقل النظام نقطة إختناق أو عنق زجاجة. تم حل هذه المشكلة على مرحلتين؛ حيث تم أولاً استخدام ناقل خاص بين الكاش الخارجي و المعالج أطلق عليه تسمية الناقل الخلفي ((Back-Side Bus (BSB) و تم بالتالي تقليل الضغط على ناقل النظام ((System Bus (Front-Side Bus). و في مرحلة أخرى، عندما سمحت تكنولوجيا المعالجات بذلك، تم نقل الـ L2 Cache نفسه إلى داخل المعالج. و بنفس الطريقة ظهر الـ L3 Cache خارجياً في البداية ثم أصبح داخلياً بعد ذلك.

عند بداية ظهوره كان الـ L1 Cache موحداً (Unified)، و لكن مع تطور تكنولوجيا المعالجات و ظهور المعالجات متعددة وحدات التنفيذ (Superscalar Machines)، مثل الـ Pentium 4 و الـ PwerPC، و التي تعتمد على التنفيذ المتوازي و الإلتقاط المسبق للتعليمات، أصبح هناك ميل نحو استخدام الكاش المقسوم (Split Cache)، حيث يتم تقسيم الـ L1 Cache إلى L1 Instruction Cache و L1 Data Cache. و ميزة الكاش المقسوم هنا هو إلغاء التنافس على الدخول للكاش ما بين وحدة الإلتقاط/فك التشفير (Fetch/Decode Unit) و وحدة التفيذ (Execution Unit). فوحدة الإلتقاط/فك التشفير هي المسؤولة عن إلتقاط التعليمات (Instructions) من الكاش و فك شفرتها، أما وحدة التنفيذ فتحتاج للرجوع للكاش لإلتقاط البيانات (Data) المطلوبة أثناء تنفيذ تعليمة معينة. فعند

حاجة كلا الوحدتين لدخول الكاش الموحد يتم إعطاء الأولوية لوحدة التنفيذ و تأخير طلب وحدة الإلتقاط. أما في حالة الكاش المقسوم فلكل من الوحدتين الكاش الخاص بها و لا يوجد أي تنافس أو تأخير.

الشكل التالي يوضح تنظيماً مبسطاً لمعالج Pentium 4 يوضح المستويات المختلفة من الكاش:



Multicore Organization