Elements of Cache Design

سنقدم نظرة عامة على الـ cache design parameters و بعض النتائج النموذجية.

الـ cache احيانا تشير الى high-performance computing (HPC) .

الـ HPC يتعامل مع اجهزة كمبيوتر العملاقة وبرامجها وخاصة التطبيقات العلمية التي تتضمن كميات كبيرة من البيانات و vector and matrix computation و parallel algorithms .

الـ Cache design يختلف في التصميم لدى cache عن المنصات الاخرى.

والباحثين وجدو ان تطبيقات HPC تعمل بشكل سيء على بنيات الكمبيوتر التي تستخدم cache .

لكن بعض الباحثين الاخرين وجدو ان من المفيد و يحسن الاداء عند اضافة cache في التسلسل الهرمي واستغلال البرامج لذاكرة cache .

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

العناصر الاساسية لدى cache في الصورة.

Cache Addresses

تدعم العديد من المعالجات الذاكرة الافتراضية (virtual memory) سنتوجه اليه في الفصول القادمة.

لكن بشكل مبسط الذاكرة الافتراضية هي ذاكرة تسمح للبرنامج بمعالجة الذاكرة من وجهة نظر منطقية بغض النظر عن الحجم الذاكرة الرئيسية المتوفرة فعليا.

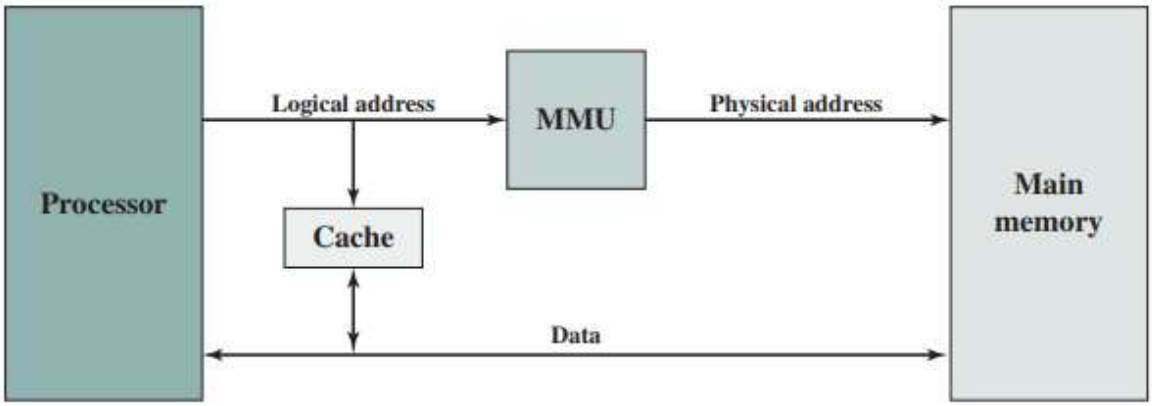
عند استخدام الذاكرة الافتراضية تحتوي على حقول العناوين بتعليمات الجهاز على العناوين الافتراضية.

بالنسبة للقراءة والكتابة من الذاكرة الرئيسية تقوم وحدة hardware memory management (MMU) بترجمة كل عنوان افتراضي (virtual address) الى عنوان فعلي (physical address) في الذاكرة الرئيسية.

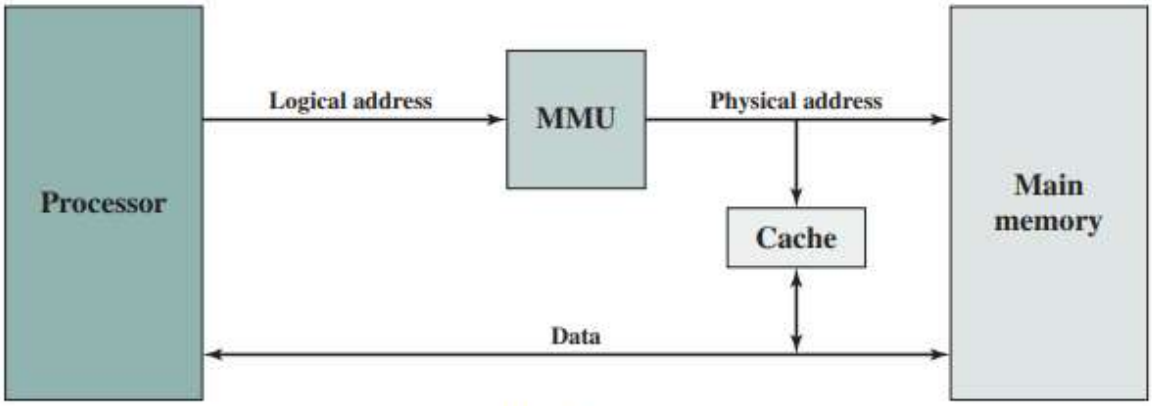
عند استخدام العناوين الافتراضية قد يختار مصمم النظام وضع ذاكرة cache بين المعالج وبين وحدة MMU او بين الذاكرة الرئيسية و MMU .

يجب ان يتم مسح cache بالكامل باستخدام كل application context switch او يجب اضافة وحدات بت اضافية الى كل سطر من cache لتحديد مساحة العنوان الافتراضي الذي يشير الى هذا العنوان.

موضوع الـ cache يعتبر معقد احنا راح نكتفي في الاساسيات.



(a) Logical cache



(b) Physical cache

الـ logical cache معرفة بـ virtual cache تقوم بتخزين البيانات باستخدام العناوين الافتراضية.

يصل المعالج الى Cache مباشرة دون المرور عبر MMU .

اما الـ physical cache تقوم بتخزين البيانات باستخدام الـ main memory physical addresses .

احدى المزايا الواضحة لـ logical cache هي ان سرعة وصول الى cache اسرع من وصول الـ cache physical لان الـ cache ممكن تستجيب قبل ما تقوم الـ MMU بترجمة العناوين.

العيب بالذاكرة الافتراضية معظم هاذي الانظمة تزود كل تطبيق بنفس مساحة عنوان الذاكرة الافتراضية.

اي كل تطبيق يرى في الذاكرة الافتراضية انه يبدأ عند عنوان 0 .

وبتالي هذا يشير الى نفس العنوان الافتراضي في تطبيقين مختلفين الى physical addresses مختلفين.

Cache Size

هو حجم الـ cache .

نود ان يكون حجم الـ cache صغير بما يكفي بحيث يكون متوسط التكلفة لكل بت وهناك دوافع اخرى يمكنك البحث عنها.

كلما كانت الـ cache اكبر زاد عدد البوابات المشاركة في معالجة addressing the cache .

والنتيجة هي ان ذاكرات التخزين المؤقت الكبيرة تميل الى ان تكون ابطأ قليلا من الـ cache الصغيرة في الحجم.

حتى عندما يتم بناؤها باستخدام نفس التقنية integrated circuit ووضعها نفس المكان على شريحة و الـ circuit board .

كما ان منطقة الشريحة واللوحة المتوفرة تحد من حجم cache .

بعض احجام الـ cache لمختلف المعالجات.

Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16–32 kB	—	—
PDP-11/70	Minicomputer	1975	1 kB	—	—
VAX 11/780	Minicomputer	1978	16 kB	—	—
IBM 3033	Mainframe	1978	64 kB	—	—
IBM 3090	Mainframe	1985	128–256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256–512 kB	—
PowerPC 601	PC	1993	32 kB	—	—
PowerPC 620	PC	1996	32 kB/32 kB	—	—
PowerPC G4	PC/server	1999	32 kB/32 kB	256 kB to 1 MB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 kB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/server	2002	32 kB	256 kB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB
Intel Core i7 EE 990	Workstation/ server	2011	6 × 32 kB/ 32 kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ server	2011	24 × 64 kB/ 128 kB	24 × 1.5 MB	24 MB L3 192 MB L4

Mapping Function

نظرا لوجود عدد اقل من الـ cache lines مقارنة بـ main memory blocks يلزم وجود خوارزمية لـ mapping main memory blocks في الـ cache lines .

ونحتاج وسيلة لتحديد كتلة الذاكرة الرئيسية التي تشغل الـ cache line .

يحدد الـ main function كيفية تنظيم الـ cache .

يمكن استخدام ثلاثة تقنيات : direct, associative, and set-associative .

سنستكشف كل وحده لوحدها. ننظر الى الهيكل العام ثم المثال.

بالنسبة للحالات الثلاثة يتضمن المثال :

- تحتوي الـ cache على 64kb .
- يتم نقل البيانات بين الذاكرة الرئيسية والـ cache كـ blocks تبلغ مساحتها 4 byte .

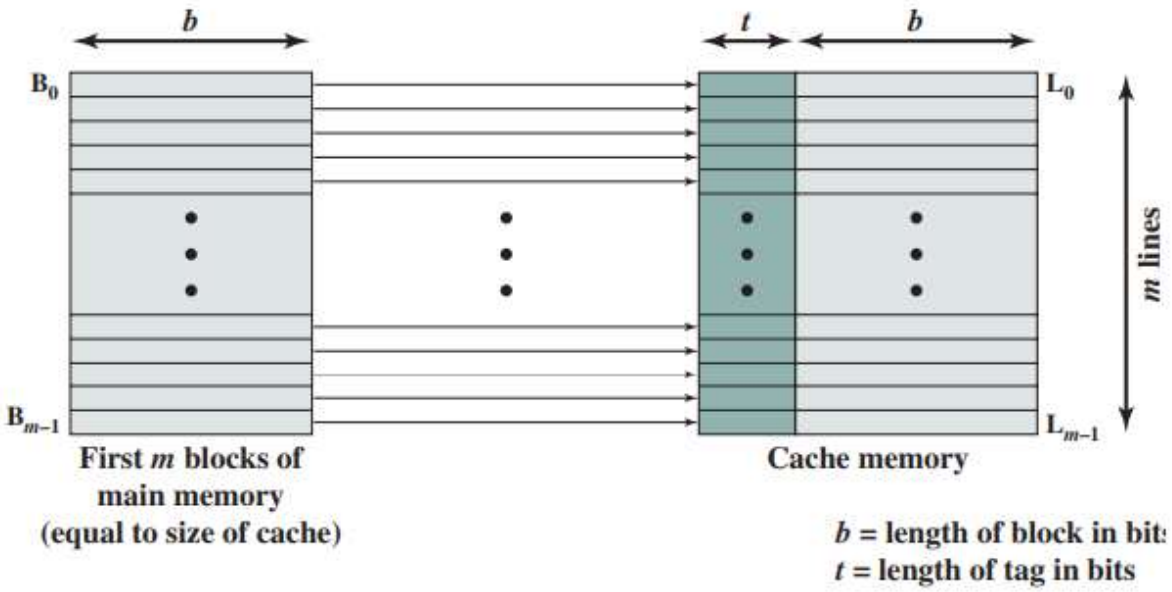
نبدأ بـ direct mapping يعتبر ابسط تقنية ممكن تتبعها تقوم بتعيين كل كتلة من الذاكرة الرئيسية في سطر الـ cache يتم التعبير عن التعين كـ :

$$i = j \text{ modulo } m$$

الـ i هو الـ cache line number .

الـ j هو الـ main memory block number .

الـ m هو الـ number of cache lines in the cache .



(a) Direct mapping

توضح هاذي الصورة mapping لكتل m الاولى من الذاكرة الرئيسية.

يتم عمل mapping لكل كتل main memory في سطر واحد فريد من cache .

ويتم تعين (map) الكتل m التالية من الذاكرة الرئيسية الى الـ cache بنفس الطريقة واي block B_m من الذاكرة الرئيسية يتم تعينه في L_0 line لحد L_{m-1} .

نفترض عندنا main memory 128 bytes و cache 32 bytes .

والـ block size is 8 bytes .

كيفية حساب الـ Blocks Main Memory عن طريق :

Number Of Blocks is : Memory size / Block Size

128 / 8 = 16 blocks .

ينتج ما يعادل 2^4 لكل block تمثل بـ 4 بت Block Index is تبدأ من 0000 الى 1111.

كيفية العثور عن مساحة الـ cache block.

Number of Lines is : Cache Size / Block Size

32/8 = 4 Lines .

هذا يعني 2^2 هذه يعني نريد 2 بت للـ cache index من 00 الى 11.

ويتم ترتيبها على البت الاقل اهميه LSB مع الاقل اهمية مثال 0000 و 0100 و 1100 يعتبرو اقل اهمية في الـ Main Memory و الـ cache الـ 00 تعتبر اقل اهمية ويمكن الـ 00 حمل 0000 و 0100 و 1100.

ونفس الموضوع يكرر مع 01 .

لكن الـ CPU لازم يكون عندو الية لفهم اي من هاذي الكتل موجودة في 1 line او 0.

يمكنها الفهم من خلال الـ Tag .

الـ Tag size is : Block Index – Cache Index

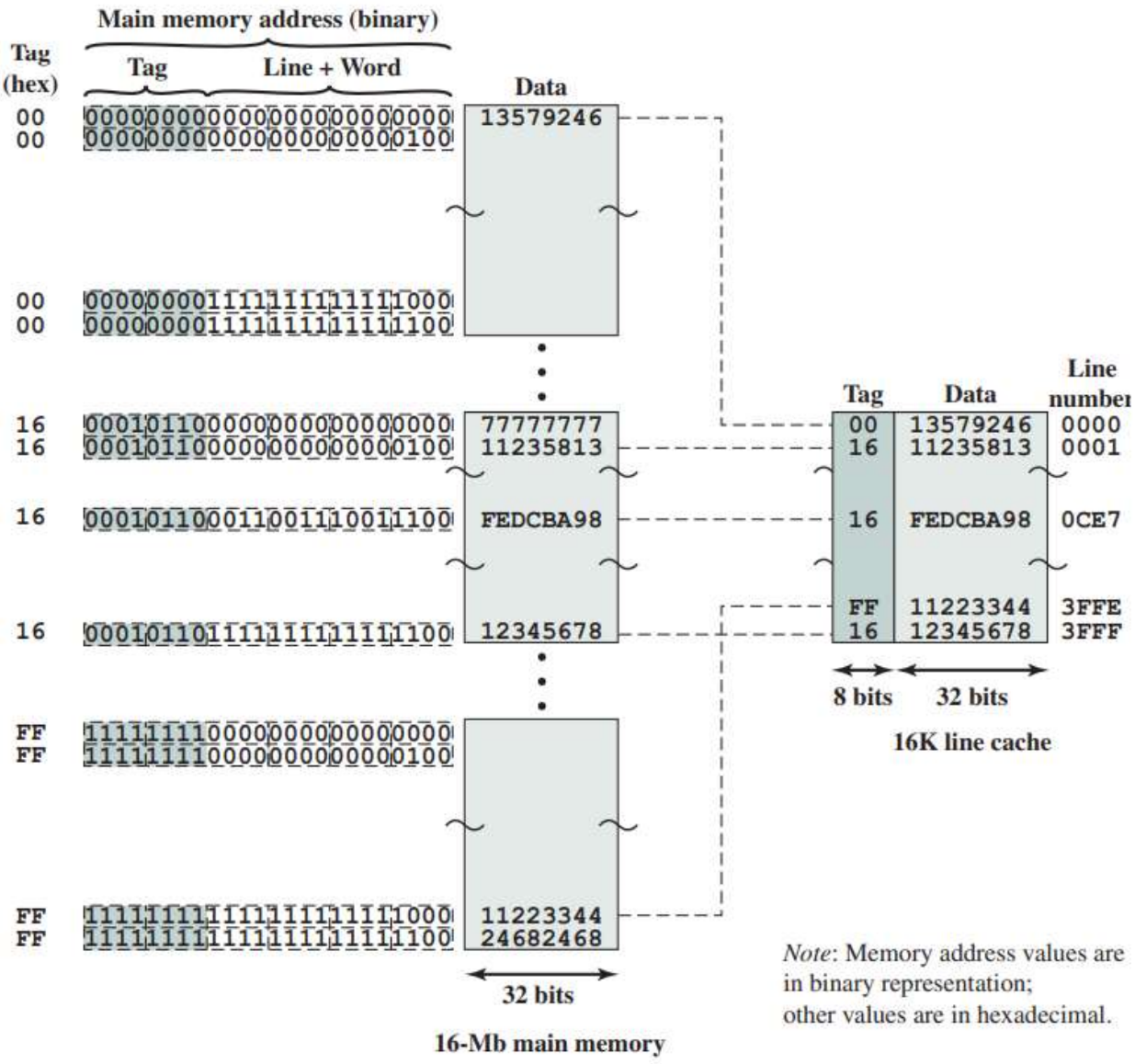
BI = 4

CI = 2

2 = 2 – 4 . نحتاج 2 بت للـ Tags بناء على الـ MSB ان كان الـ MSB 00 يكون الـ Tag 00 وان كان 11 يكون 11 وان كان 01 يكون 01 الخ.

- Address length $= (s + w)$ bits
- Number of addressable units $= 2^{s+w}$ words or bytes
- Block size = line size $= 2^w$ words or bytes
- Number of blocks in main memory $= \frac{2^{s+w}}{2^w} = 2^s$
- Number of lines in cache $= m = 2^r$
- Size of cache $= 2^{r+w}$ words or bytes
- Size of tag $= (s - r)$ bits

صورة توضح :



التاثير للـ mapping هو انه يتم mapping لكتل الذاكرة الرئيسية لأسطر الـ cache :

Cache line	Main memory blocks assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
⋮	⋮
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

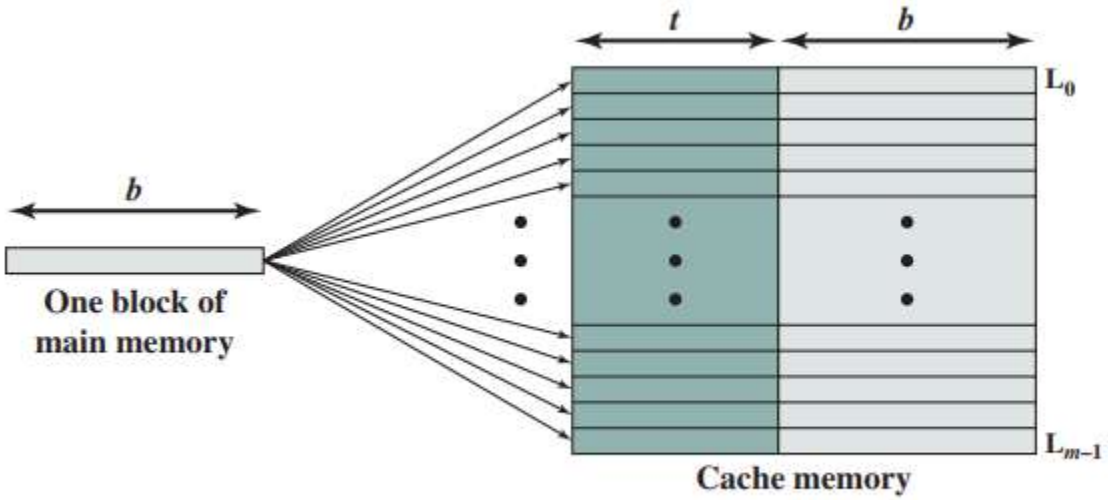
استخدام جزء من العنوان كرقم سطر يوفر تعيينا فريدا لكل كتلة من الذاكرة الرئيسية في الـ cache .

عندما بالفعل يتم قراءة الـ block في السطر المخصص لها من الضروري وضع علامة على البيانات لتمييزها عن الكتل الاخرى التي يمكن ان تناسب مع هذا السطر.

Associative mapping

النوع الثاني من الـ Mapping النوع هذا يتغلب على العيوب التي كانت موجودة في الـ Direct Mapping من خلال السماح لأي كتلة من الـ main memory يتم تحميلها في اي سطر من الـ cache .

صورة توضح ذالك :



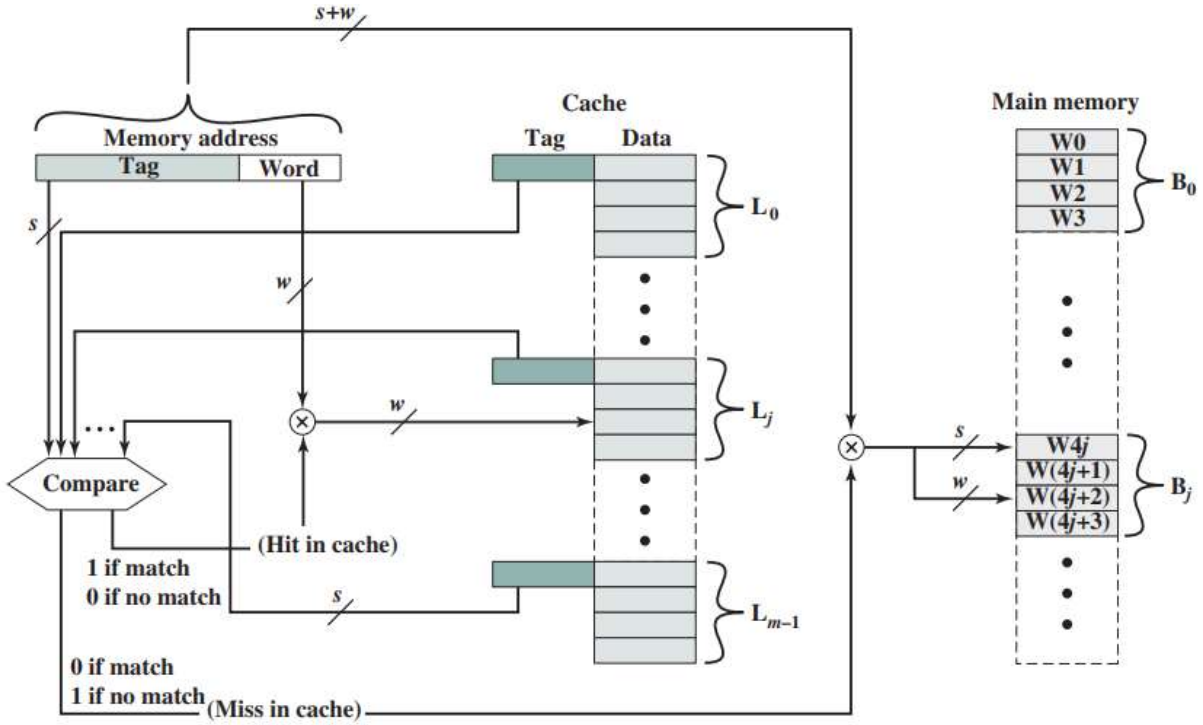
(b) Associative mapping

في هاذي الحالة يفسر منطق التحكم في الـ cache (cache control logic) على انها عناوين ذاكرة كعلامة (Tag) وحقل كلمة (Word field) .

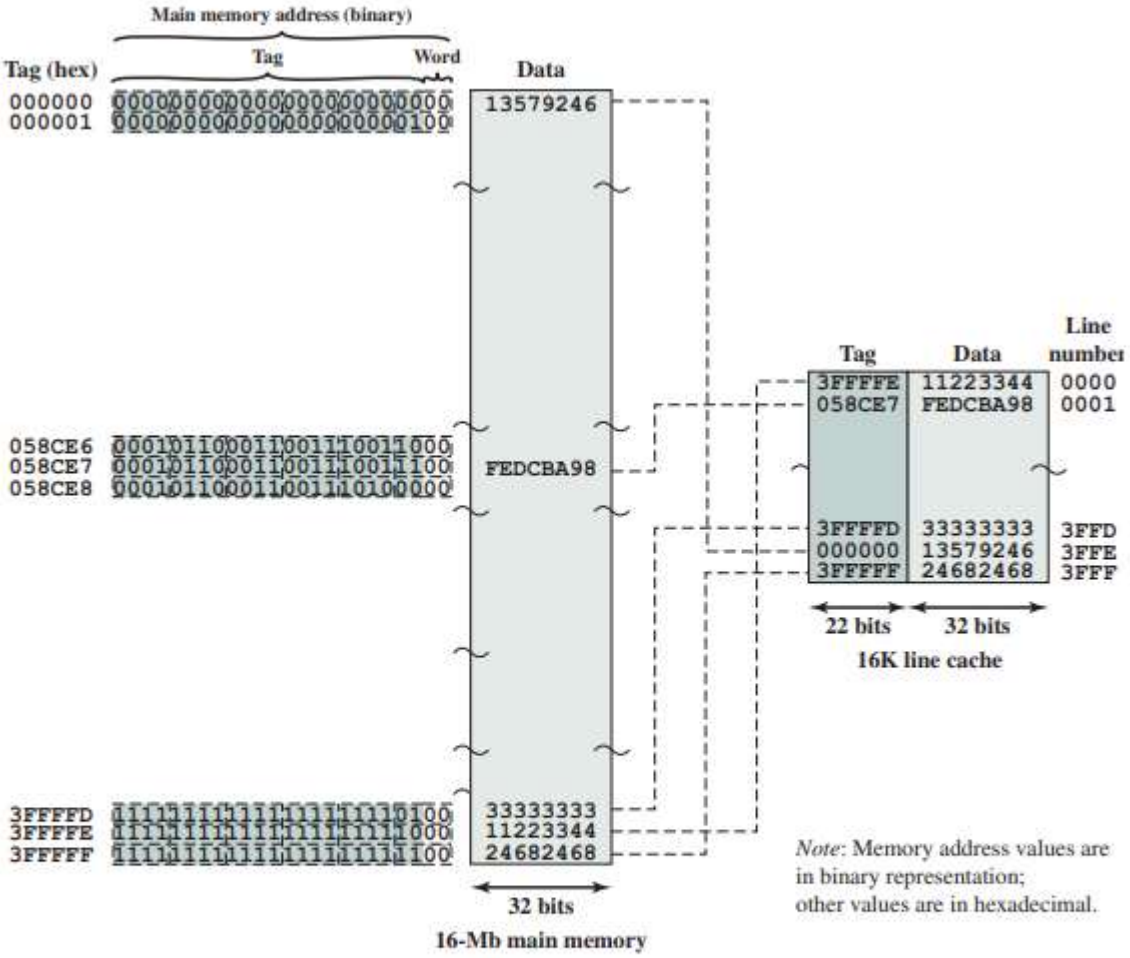
الـ Tag field يحدد بشكل فريد الكتلة من الذاكرة الرئيسية.

لتحديد ما اذا كانت الكتلة الموجودة في الـ Cache يجب ان يقوم الـ cache control logic بفحص علامة (Tag) كل سطر في نفس الوقت حتى يجد التطابق.

الصورة توضح ذالك :



مثال :



توضح هاذي الخارطة الـ associative mapping .

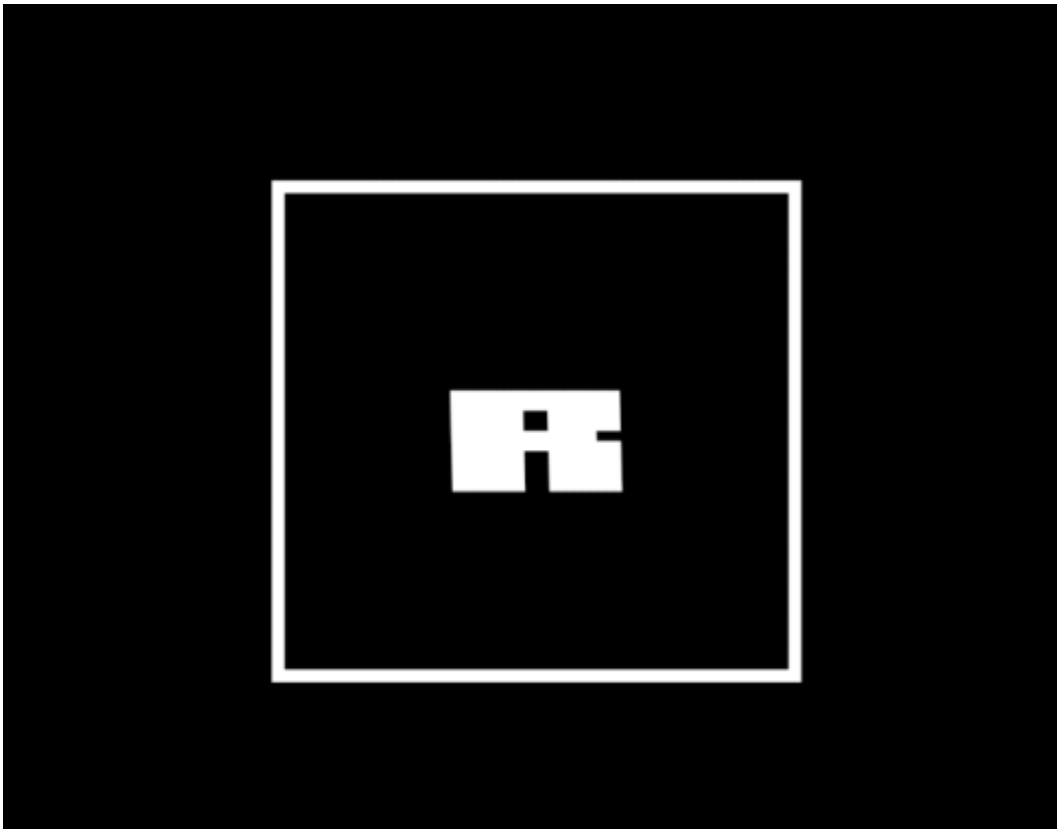
لو تلاحظ ايضا لا يوجد حقل يتوافق مع رقم الـ line بحيث لا يتم تحديد عدد الاسطر في ذاكرة التخزين المؤقت بواسطة تنسيق العنوان.

والعيب هنا فقط الدوائر المعقدة المطلوبة لفحص الـ tags و جميع cache lines بتوازي.

- Address length $= (s + w)$ bits
- Number of addressable units $= 2^{s+w}$ words or bytes
- Block size $=$ line size $= 2^w$ words or bytes
- Number of blocks in main memory $= \frac{2^{s+w}}{2^w} = 2^s$
- Number of lines in cache $=$ undetermined
- Size of tag $= s$ bits

Set-associative Mapping

على ما اعتقد والله اعلم لم اعطي حق العلم في هذا الدرس فهذا مصدر جيد جدا لتعلم بشكل افضل من هاذي الورقة :
https://www.youtube.com/watch?v=4i_UQ2j2ynQ&list=PLYzc3veT5szLyYxJB . MZ9cIRXD3Xv4Ceur



Twitter : https://twitter.com/dr_retkit

YouTube : https://www.youtube.com/@retkit1823