Memòria caché (1)

Justificació de l'ús de cachés

La memòria caché és un dels tipus de memòries que fem servir dintre de la jerarquia de memòria d'un computador.

El primer que podem pensar és:

– Per què hem de tenir més d'un tipus de memòria dintre d'un computador?

La resposta és molt senzilla:

– No existeix actualment el dispositiu de memòria ideal.

Les característiques que hauria de tenir una memòria ideal són:

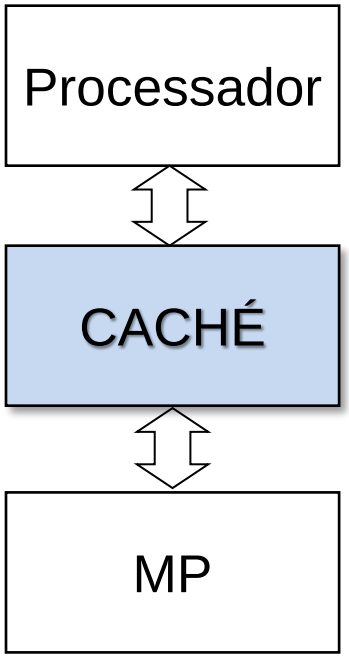
- Ràpida com el processador*.
- Capacitat molt gran en poc espai (alta densitat d'integració).
- Poder retenir les dades quan no hi ha energia.
- Poder Escriure i Llegir a la mateixa velocitat.
- Econòmica.

Tot i que hi ha memòries amb algunes d'aquestes característiques, no hi ha cap amb totes. A més, hi ha propietats que solen ser antagòniques (memòries ràpides solen tenir poca capacitat, i no solen ser econòmiques...).

Situació de la Caché a la jerarquia de memòria

La funció bàsica de les memòries caché és subministrar informació el més ràpid possible al processador (Instruccions i/o dades).

Per tant, han de ser memòries molt ràpides i molt properes al processador.



Caché a la jerarquia de memòria

- La seva situació està entre el processador (just per sota del nivell de registres d'aquest) i la memòria principal.
- Hem de ser conscients que a la caché tindrem sempre informació que és un duplicat de al que hi ha a la MP.
- Degut a que és molt més petita que la MP, serà molt important gestionar quina informació hi haurà a cada instant (convindria que fos la que li fa falta al processador).

Beneficis que aporta la Caché al sistema

Els beneficis que proporciona la caché poden ser:

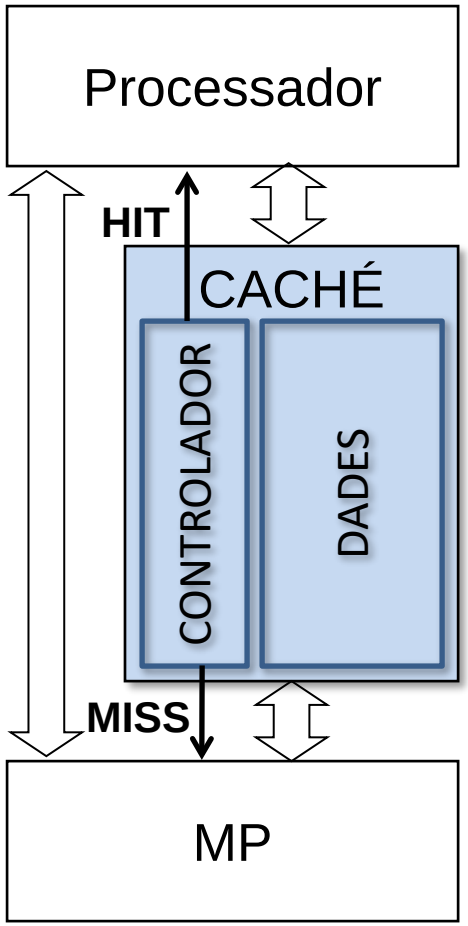
1. Subministrar instruccions i/o dades al processador al ritme que els necessita (més ràpid que la MP). Això ha estat crucial sobretot amb la implementació de *pipelines* als processadors.

2. Permetre que els processadors puguin escriure els resultats de les operacions al ritme apropiat.

3. En moltes configuracions la utilització de cachés permet disminuir el tràfic d'informació a través dels busos de sistema.

Principi de funcionament de la caché dins el sistema

Una possible implementació d'un sistema amb caché podria ser*:



El principi de funcionament és aquest:

- El processador, a través del sistema de busos, demana una informació (instrucció o dada) al sistema de memòria.
- Si la dada es troba a la caché es produeix un HIT, i la caché subministra la dada al processador.
- Si la dada no es troba ala caché es produeix un MISS, la memòria llavors serà qui entregui la dada.

Principi de funcionament de la caché dins el sistema

La clau per què un sistema de caché sigui eficaç és senzillament que el nombre de HITs sigui molt alt (i per tant el MISS molt baix).

A un sistema ben ideat, el nombre de HIT (el que es denomina "*hit rate*") ha de ser del 90% o més.

Això depèn de molts factors: configuració del sistema, organització interna de la caché, dels algoritmes de reemplaçament de dades...

I el principis bàsics en que es basa són els de sempre:

- Localització Espacial: La informació que es necessita en un futur proper durant la execució d'un programa sol estar a prop de la que es fa servir en aquest moment. Ex: Vectors, matrius o blocs de dades, instruccions consecutives.
- Localització Temporal: Els programes acostumen a fer servir informació recent. Ex: Bucles, subrutines, blocs de dades...

Accés Associatiu a la informació

Per una part, la manera en que el processador cerca la informació al sistema (amb l'adreça que té la informació a MP) i per altra el fet que a la caché la informació és un duplicat de la que hi ha a MP, fan que l'estructura interna de la caché sigui més complicada que una memòria d'accés directe.

Dintre de la caché s'han de emmagatzemar 2 tipus d'informació:

- **Caché Data**: La informació que ens interessa (Instruccions o dades).
- **Caché TAG**: Informació sobre l'adreça que aquesta informació té a MP.

A més, la caché disposa d'una lògica "**Controlador de Caché**" que té com funcions bàsiques:

- Detectar si té o no la informació que se li demana.
- Si no la té (MISS) fer que aquesta informació es cerqui a MP.
- Saber comunicar-se amb el processador i amb la MP.

PARAMETRITZACIÓ – RENDIMENT del sistema de Caché

Un dels paràmetres més importants és el:

$$Hit\ Rate = h = \frac{número\ Caché\ Hits}{número\ peticions\ a\ memòria}$$

És un valor entre 0 i 1. En sistemes bons ha de ser de 0.9 com a mínim.

El Miss Rate es defineix com: "*Miss Rate* = 1-*Hit Rate* = 1-h"

Un altre paràmetre molt important és el *Miss Penalty* (M):

- Temps addicional que es necessita per obtenir la dada quan no es troba a caché.

PARAMETRITZACIÓ – RENDIMENT del sistema de Caché

Amb aquests paràmetres, el Temps Mig d'accés a la informació és:

$$t_{med} = h \cdot T_{ac} + (1-h)M$$

Temps dades trobades a caché

Temps dades NO trobades a caché

On T_{ac} és el temps d'accés a caché, *h* el *Hit Rate* i *M* el *Miss Penalty*.

I el guany (G) que aconseguim amb el sistema de caché serà:

$$G = \frac{Temps\ sense\ caché}{Temps\ amb\ caché}$$

Exemple de rendiment a un sistema de Caché

Suposem un sistema amb les següents característiques:

- Temps d'accés a MP siguin 10 cicles de rellotge: $t_{aMP}=10 \cdot T_C$
- Temps d'accés a caché un cicle de rellotge: $t_{ac}=T_C$
- *Miss Penalty*, $M=17 \cdot T_C$
- *Hit Rate* d'instruccions = 0.95
- *Hit Rate* de dades = 0.9

Suposem que un 30% de les instruccions fan accés a dades de memòria.

El guany és (calcularem temps de 100 instruccions sense i amb caché):

$$t_{sense\ cache} = 130 \cdot 10 \cdot T_C$$

$$t_{amb\ cache} = 100\,(0.95 \cdot T_C + 0.05 \cdot 17 \cdot T_C) + 30\,(0.9 \cdot T_C + 0.1 \cdot 17 \cdot T_C)$$

Contribució de l'accés a instruccions

Contribució de l'accés a dades

$$G = \frac{t_{sense\ cache}}{t_{amb\ cache}} = 5.04$$

RENDIMENT d'un sistema de Caché de 2 Nivells

Un sistema de caché de dos nivells vol dir que tenim dos cachés entre el processador i MP.

Quan el processador demana informació primer es cerca a la caché de nivell 1 (L1), si es troba perfecte, sinó es cerca a la caché de nivell 2 (L2) i en cas de que tampoc estigui aquí llavors la proporcionarà la MP.

Amb aquesta configuració el que aconseguim és augmentar el hit rate global del sistema de caché.

El temps mig ara serà:

$$t_{med} = h_1 \cdot T_{ac1} + (1-h_1) h_2 \cdot T_{ac2} + (1-h_1)(1-h_2) M$$

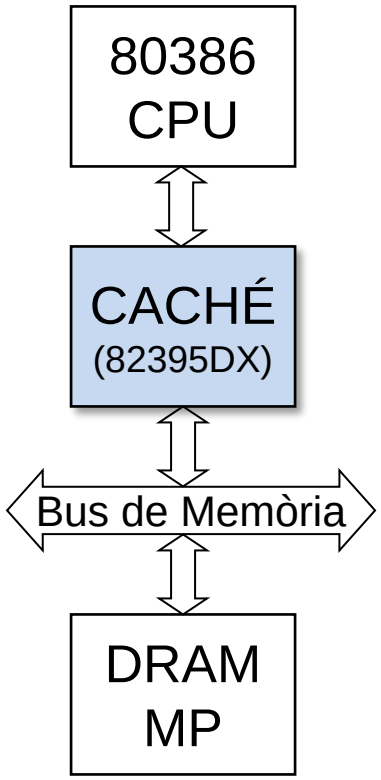
Temps dades trobades a caché 1

Temps dades trobades a caché 2

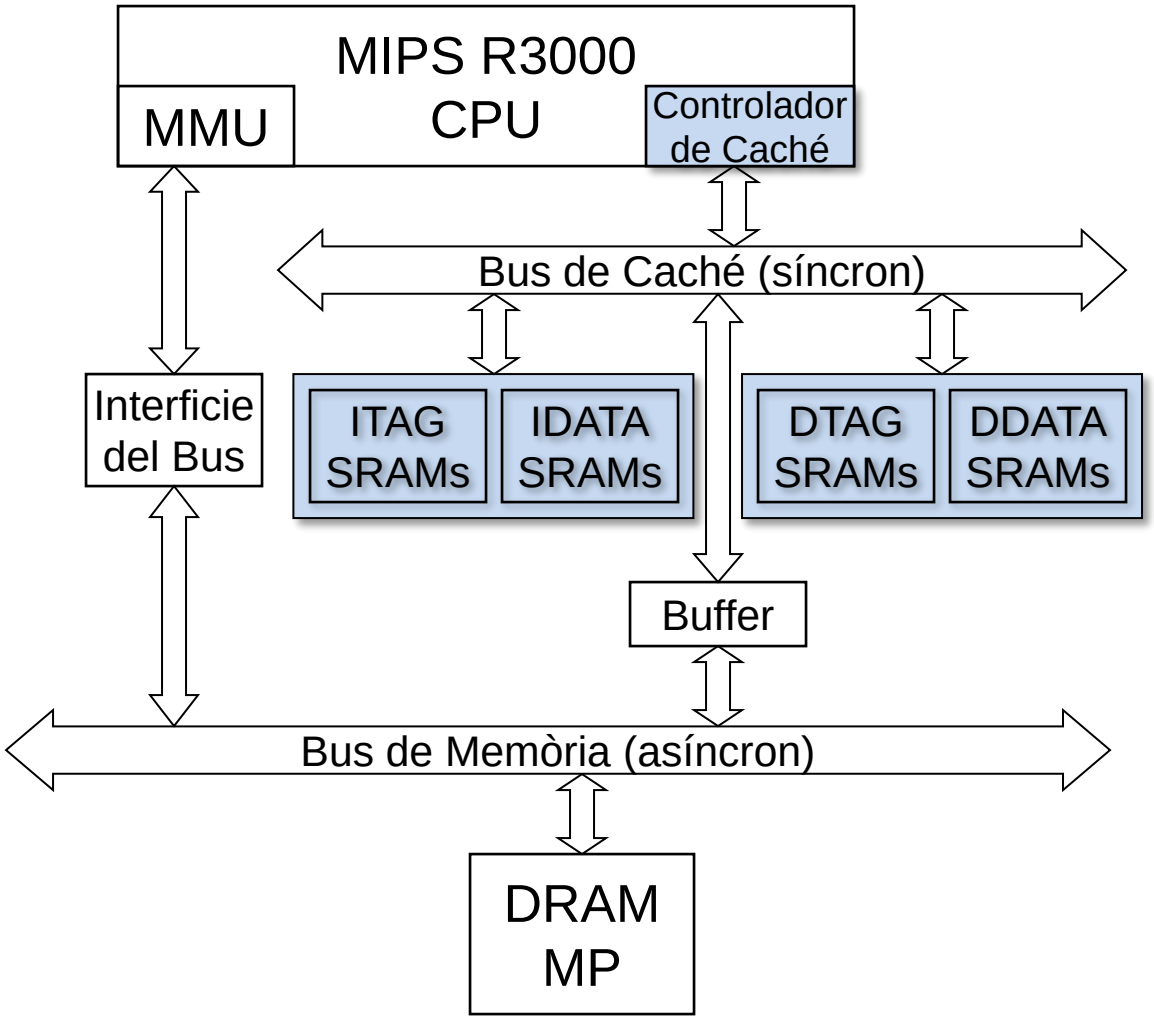
Temps dades NO trobades a caché

On: T_{aci} és el temps d'accés a caché 1 i 2; h_i el *Hit Rate de caché 1 i 2;* i M el *Miss Penalty*.

Exemples de configuracions de sistemes amb Caché

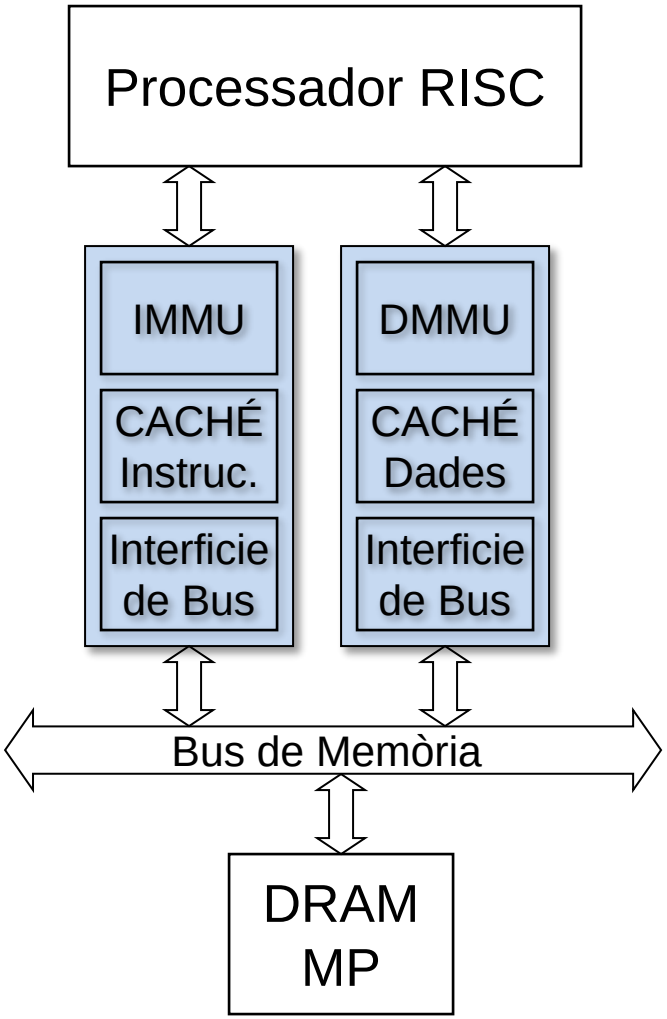


Ex. 1:Caché
Externa Unificada

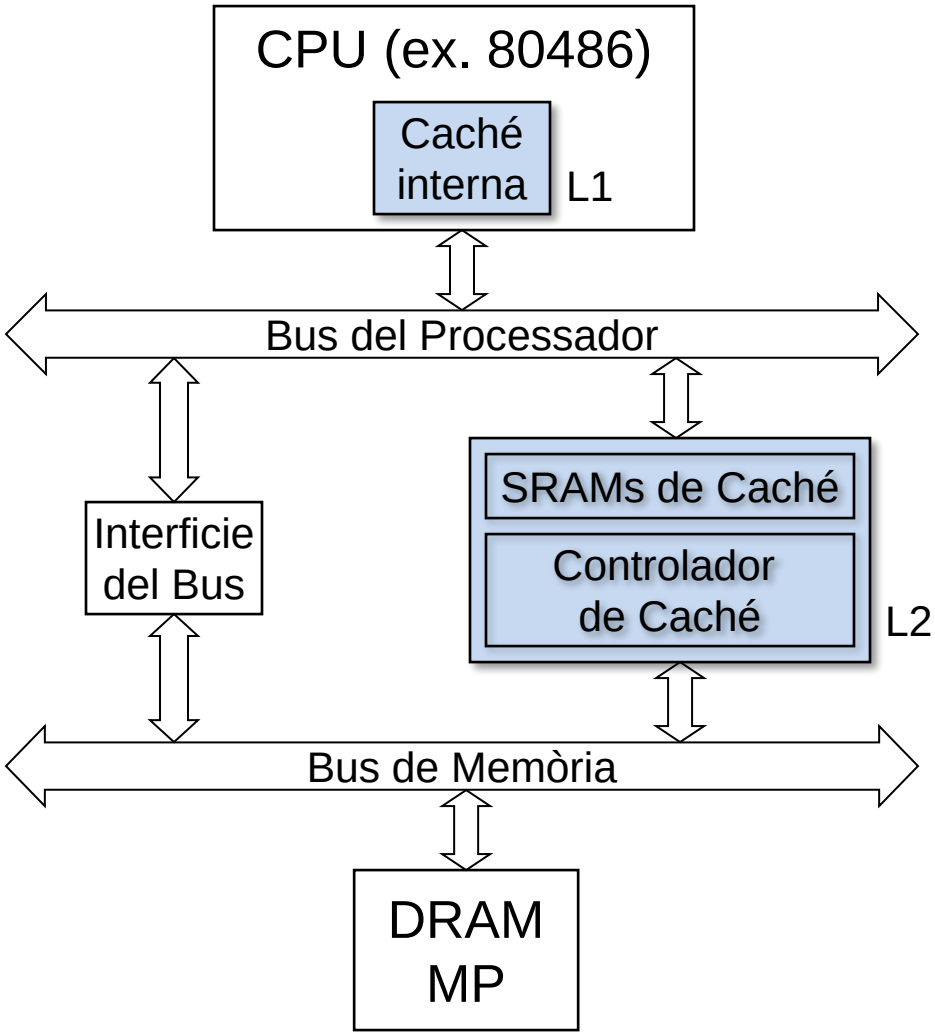


Ex. 2: Cachés Externes separades. Busos independents.

Exemples de configuracions de sistemes amb Caché

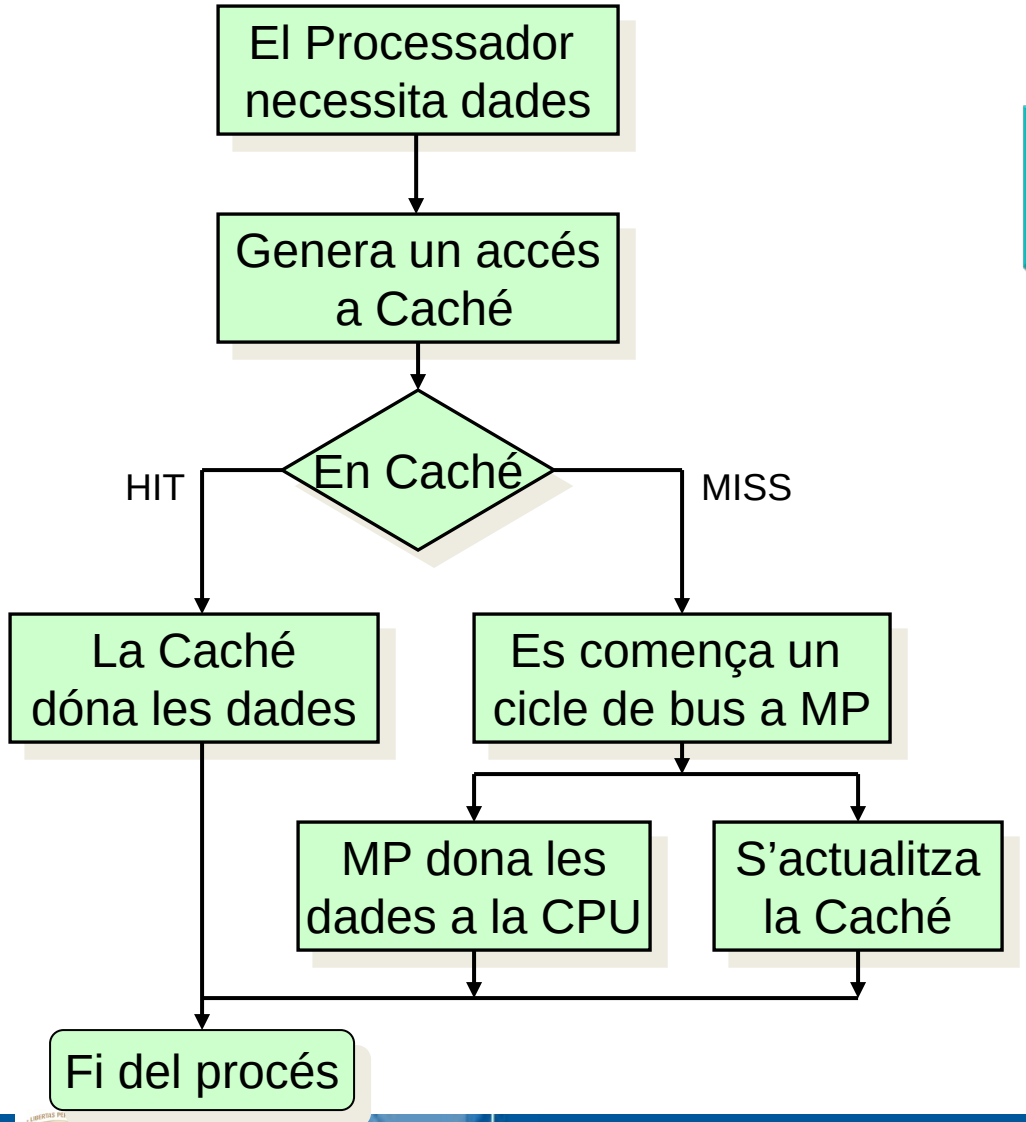


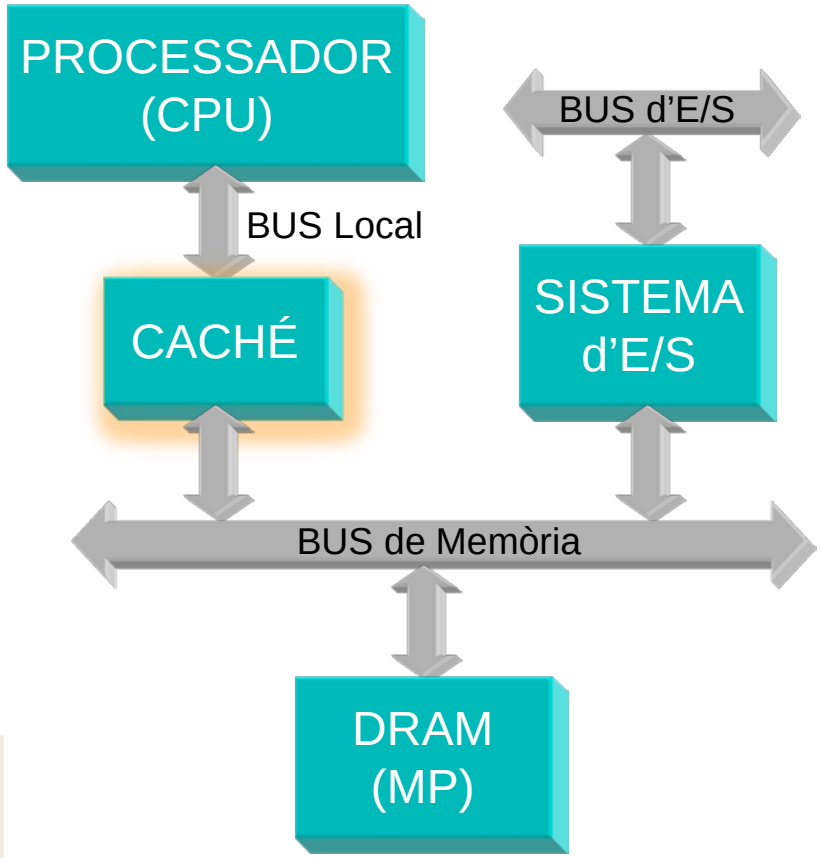
Ex.3: Cachés externes separades (arq. Harvard)



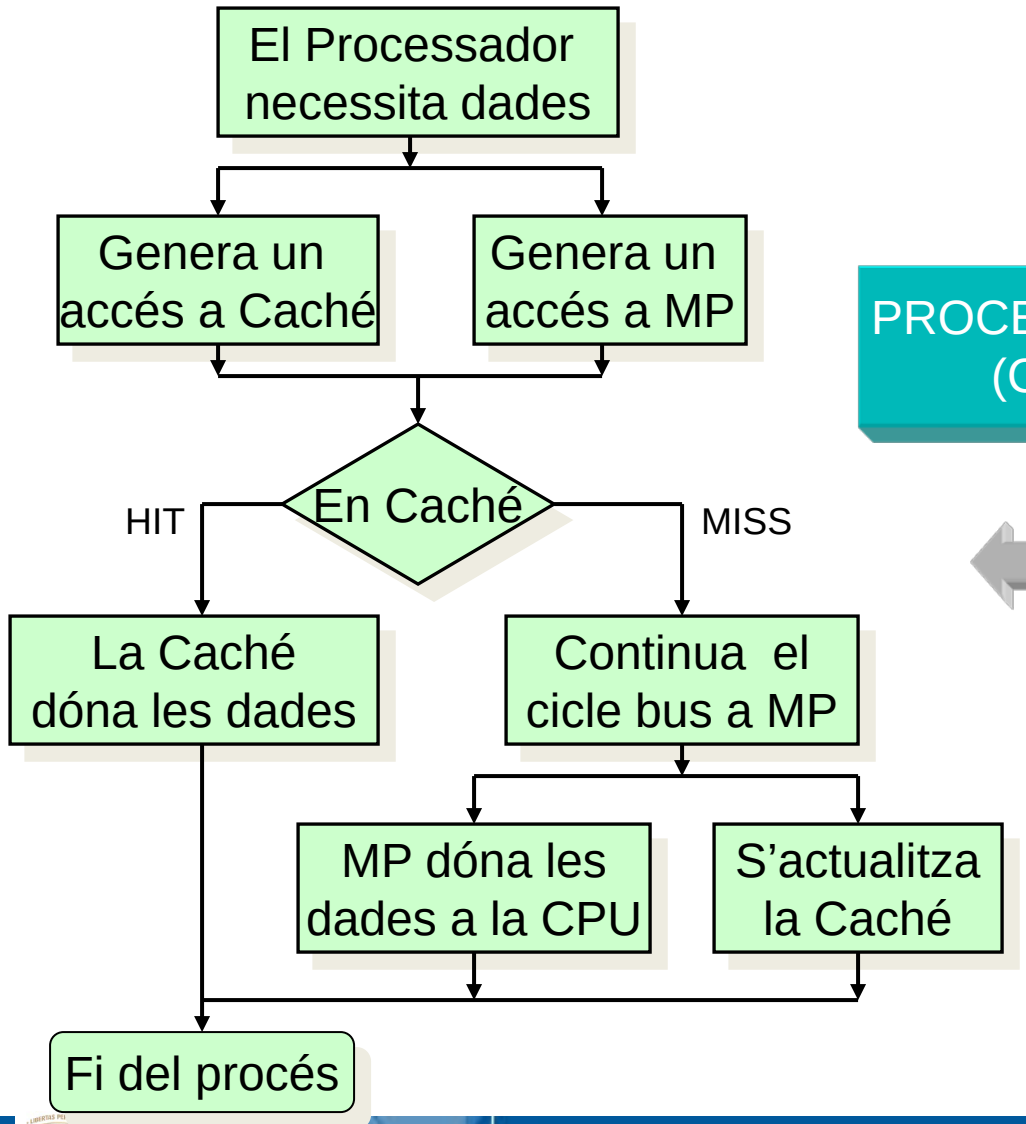
Ex.4: Cachés de nivell 1 i 2

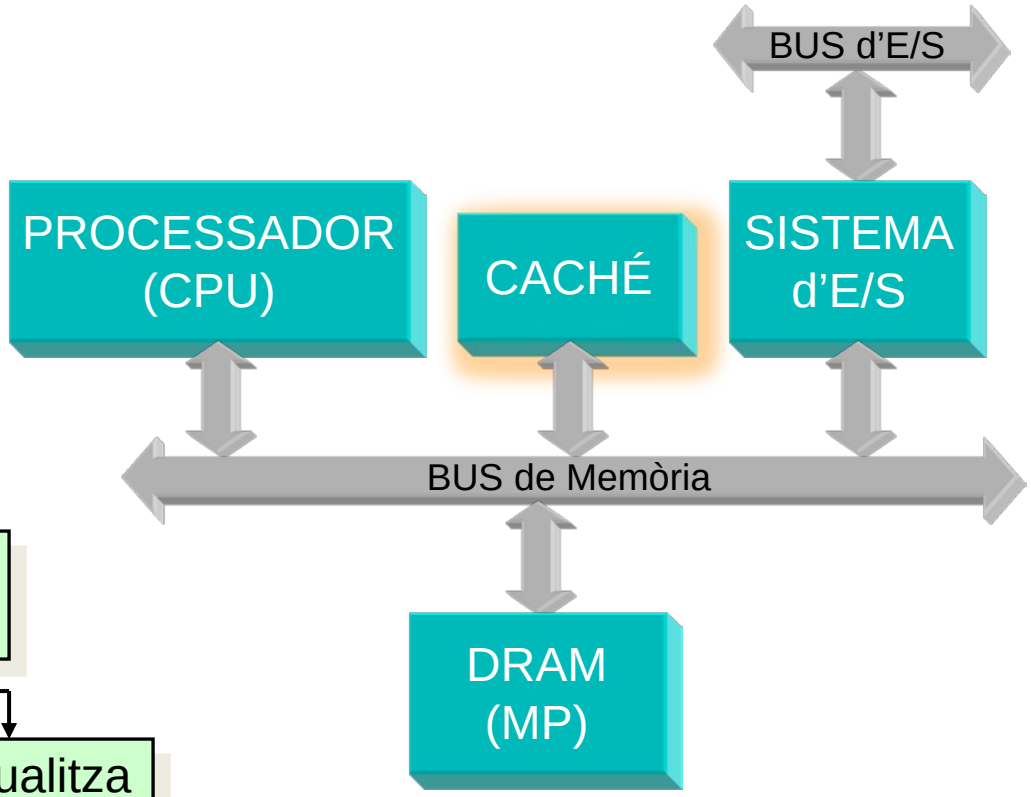
ARQUITECTURA LOOK-THROUGH





ARQUITECTURA LOOK-ASIDE





Què passa quan el processador vol escriure a Caché?

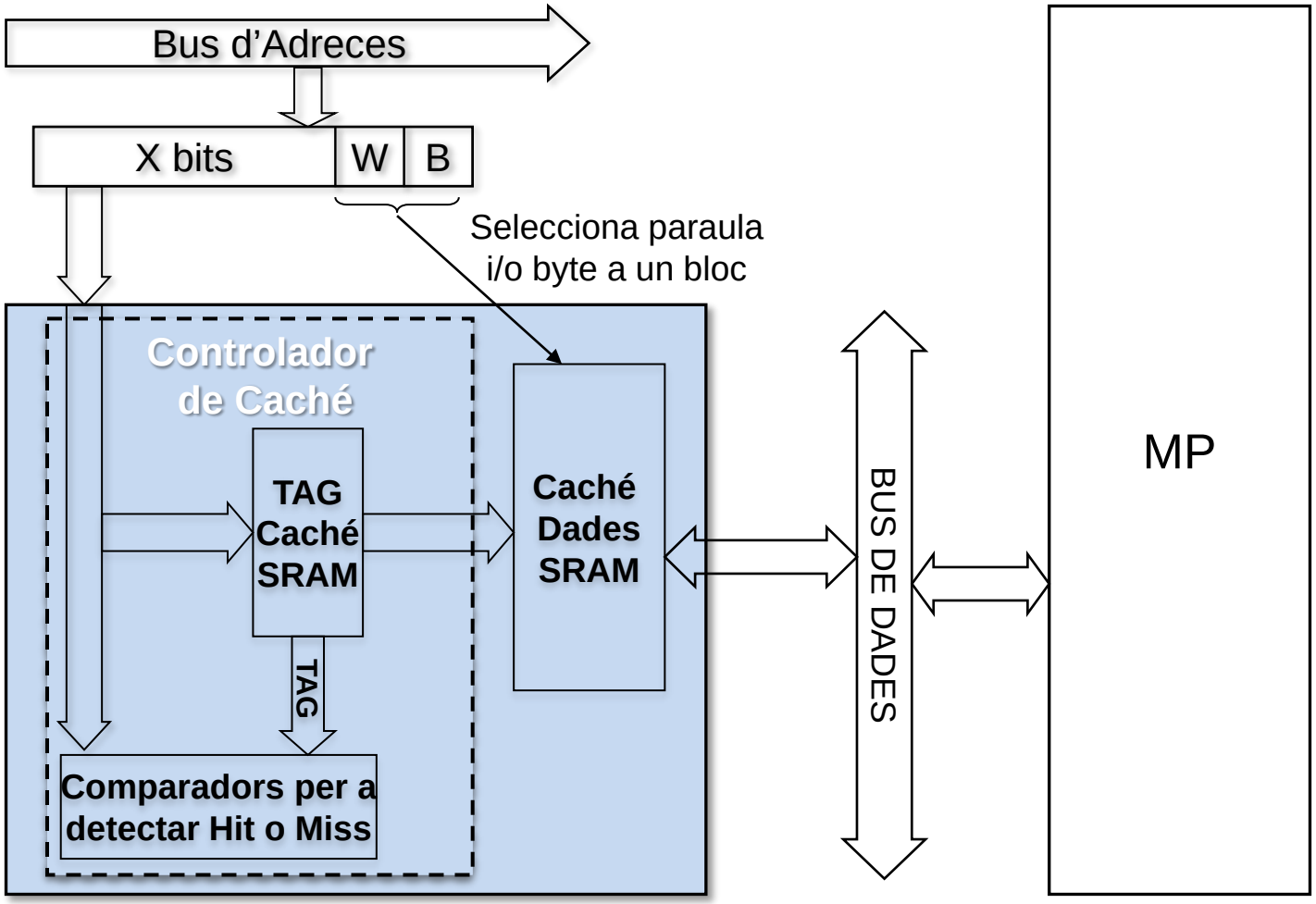
Fins ara només hem contemplat la situació en que el processador cerca informació a caché, però també podem fer servir la caché per emmagatzemar resultats d'operacions realitzades al processador.

Bàsicament hi ha dos formes d'actuar (Polítiques d'Escriptura):

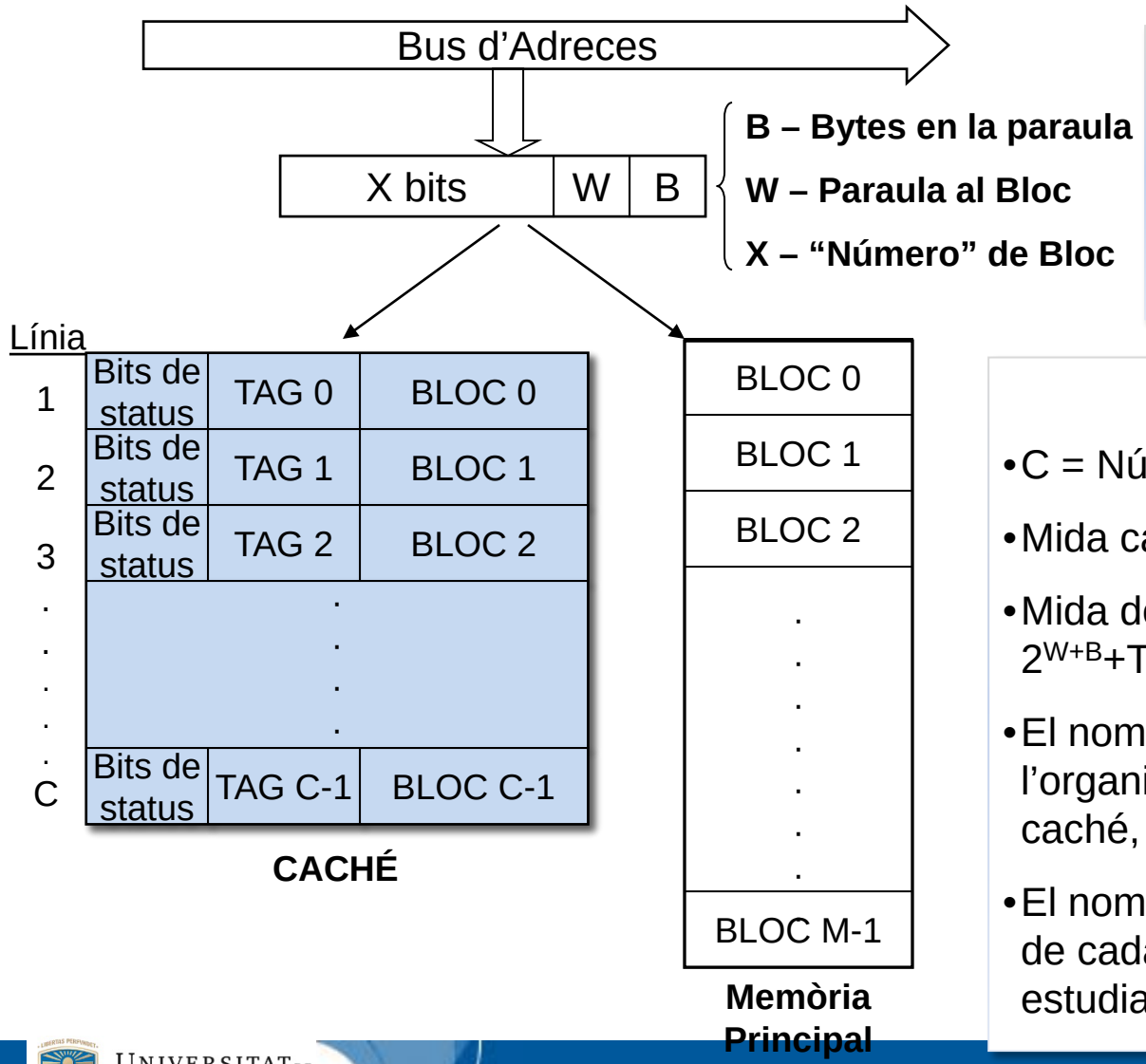
1. **WRITE-THROUGH (W-T)**: La dade s'escriu a la caché però també s'escriu a la memòria principal.
2. **WRITE-BACK (W-B)**: La dada només s'escriu a la caché

Avantatges Write-Through	Avantatges Write-Back
Facilita la Coherència de Informació	"És més ràpid"
La gestió de la informació a caché és més senzilla.	Redueix el tràfic al bus de sistema

Organització de la Memòria i la Caché en Blocs



Organització de la Memòria i la Caché en Blocs



Paràmetres Globals

- 2^B: Número de bytes per paraula
- 2^W: Número de paraules per bloc
- Mida dels Blocs: 2^{W+B}

Paràmetres Caché

- C = Número de línies de caché.
- Mida caché=C·(Mida línia)*
- Mida de les línies de caché: 2^{W+B}+TAG+"nº bits status".
- El nombre de bits X depèn de l'organització de les dades a la caché, i de la mida de les adreces.
- El nombre de bits de status depèn de cada implementació. Ja ho estudiarem després.

Organització de la Memòria i la Caché en Blocs

2^B indica el número de Bytes que hi ha a una paraula. Això és una característica del processador: Un processador de 32 bits (això indica que la seva paraula és de 32 bits) tindrà 4 Bytes i per tant $B = \log_2 4 = 2$. Que és el número de bits que es necessiten per poder diferenciar entre els 4 bytes de la paraula.

2^W indica el número de paraules que hi ha a un bloc. Això depèn de la mida de les paraules (paràmetre del processador) i dels blocs (paràmetre del nostre sistema de caché). $W = \log_2$(número de paraules a un bloc).

X=(Mida Adreça)-W-B.

Ex. Un processador de 32 bits de dades (B=2), Mida de bus d'adreces=32 i amb sistema de caché de blocs de 128 paraules (W=7): X=32-7-2=23.

Organització de la Memòria i la Caché en Blocs

Ens referirem com ORGANITZACIÓ de CACHÉ com la forma en que s'organitzen les dades dintre la caché.

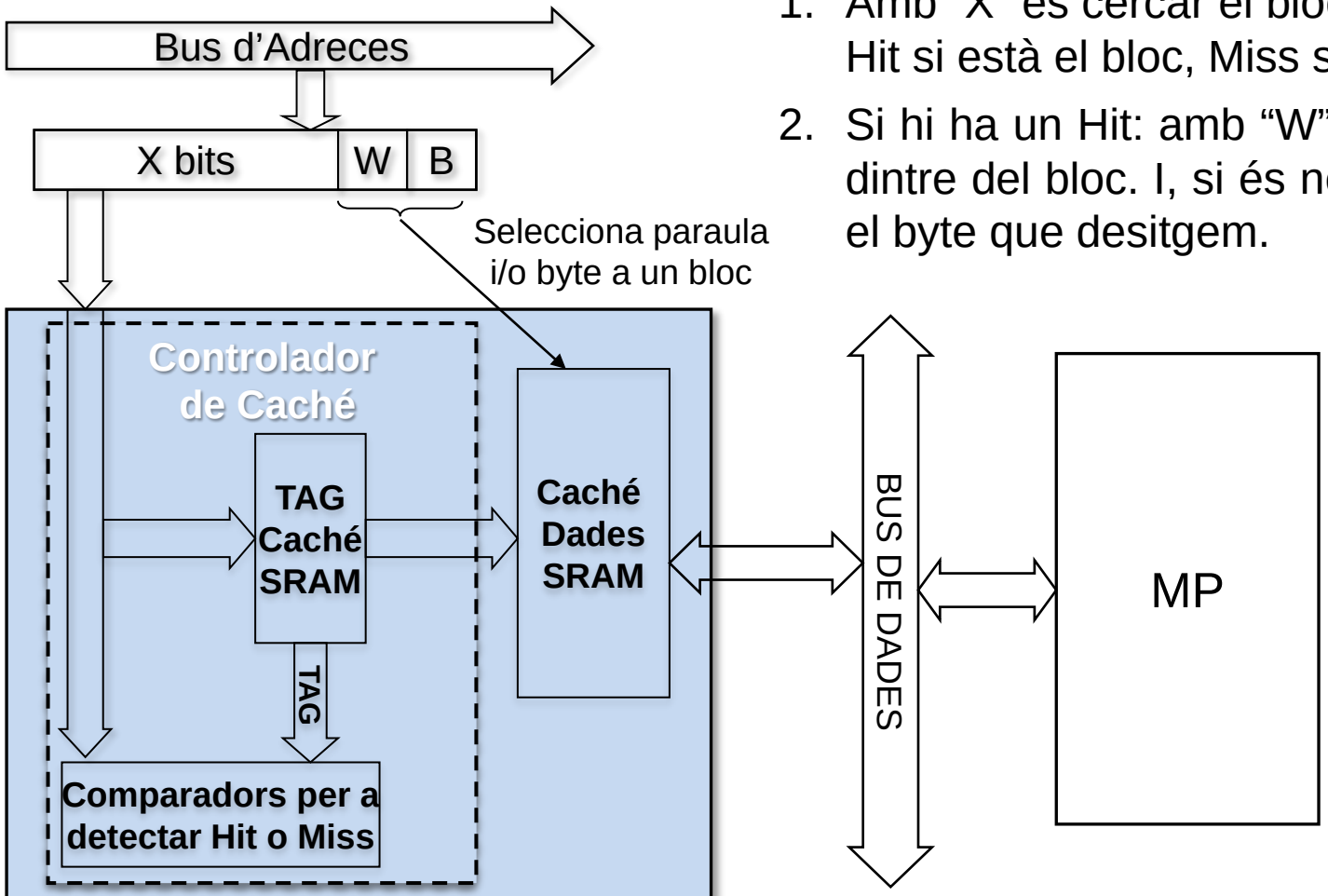
Per organitzar les dades a la caché, de manera que després sigui efectiva la seva cerca (hem de pensar que el identificador de la informació sempre és la seva adreça a MP, encara que estigui a la caché), es fa el següent:

1. La memòria principal (MP) es divideix en blocs idèntics, la mida dels quals serà 2^{W+B}.
2. Cada bloc d'aquests té un espai assignat a la caché, al que denominem LÍNIA de caché. Cada línia té una mida que és: la del bloc més un cert espai per "bits de status" i el que denominarem TAG (etiqueta), en total: 2^{W+B}+TAG+"nº bits status"
3. La mida del camp "bit de status" dependrà de cada implementació.
4. La mida del TAG depèn de com organitzem les dades a la caché i la mida de les adreces del processador.

Com es cerca una dada a caché?

Per cerca una dada a caché el procés es fa en dues parts:

1. Amb “X” es cercar el bloc on està la dada. Hit si està el bloc, Miss si no està.
2. Si hi ha un Hit: amb “W” aïllem la paraula dintre del bloc. I, si és necessari, amb “B” el byte que desitgem.



Com es cerca una dada a caché?

Com es cerca un bloc determinat a la caché? :

Les C línies de caché les agrupem en conjunts. "**S**" és el número de conjunts que fem (cada conjunt té el mateix número de línies "**K**" que anomenarem camins). **Un bloc de MP té assignat un conjunt específic de la caché**.

Els X bits d'adreces que vàrem dir que fem servir per localitzar un bloc a la caché els descompondrem en dos parts:

$$X = TAG + i$$

- **i** : ens indica a quin conjunt ha d'anar un bloc de MP a la caché: **$i=\log_2 S$**.
- **TAG** : Serveix per localitzar un bloc específic dintre d'un conjunt.

Amb els "i" bits es determina el conjunt de línies al que està assignat aquest bloc mitjançant un decodificador de "i" entrades i $S=2^i$ sortides.

Un cop sabem el conjunt al que hauria d'estar el nostre bloc, hem de veure si efectivament està a la caché. Això ho fem comparant la part de TAG de l'adreça de la informació que cerquem, amb tots els camps TAGs de les línies que hi ha al conjunt seleccionat.

CONJUNTS I CAMINS

Segons el que acabem de dir, tindrem un número de conjunts “S” cada un amb “K” línies i es complirà que:

$$C = K \cdot S$$

- **K = número de Camins**, indica a quantes línies diferents, sense cap restricció, pot anar un bloc de MP quan el fiquem a la caché.
- **S = Número de Conjunts**, indica quin grau de accés “directe” tenim al posar un bloc MP a la caché.

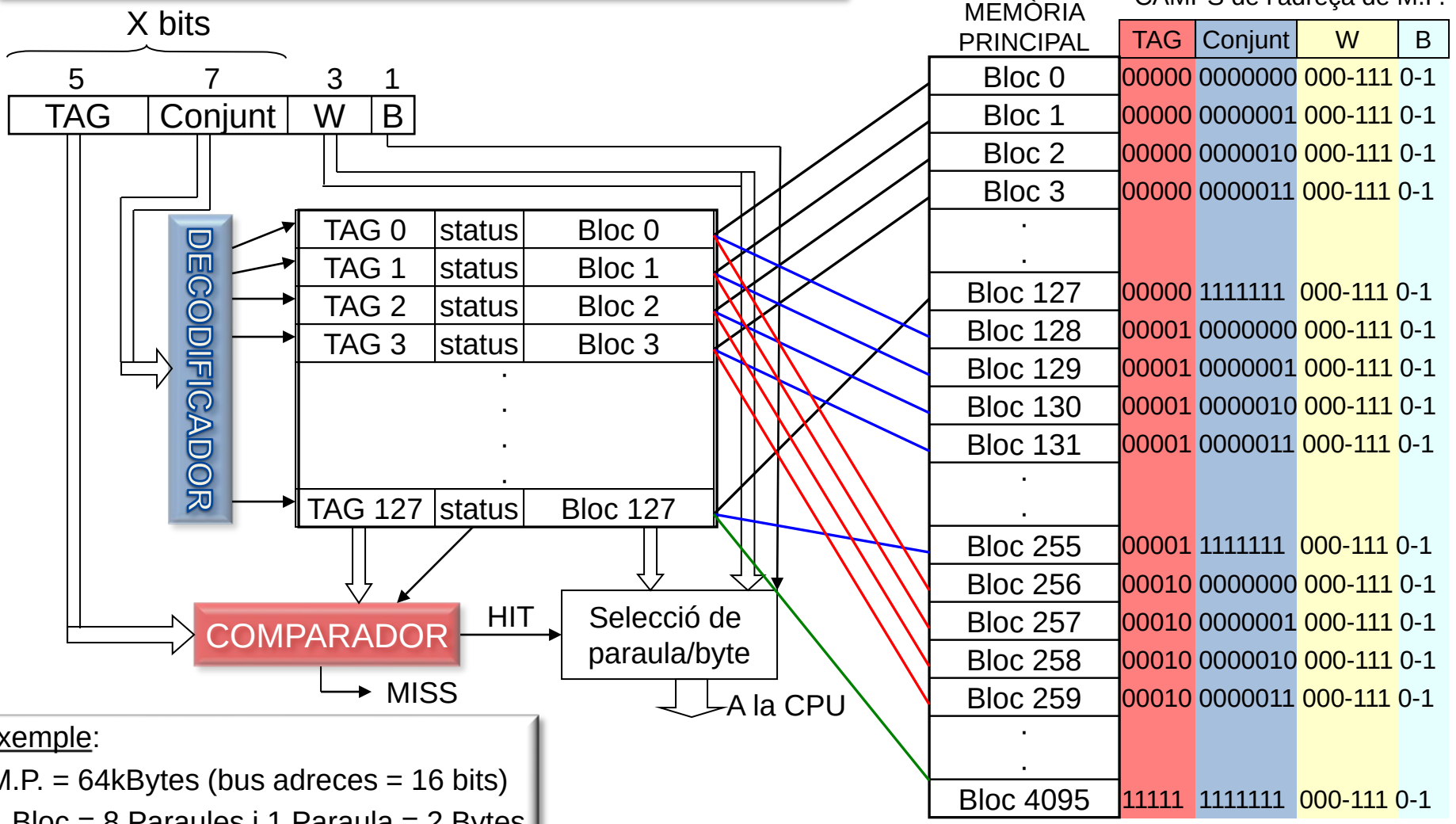
Amb aquesta definició, distingirem 3 possibles organitzacions:

- K=1, S=C : **Caché de Mapejat Directe**.
- K=C, S=1 : **Caché Completament Associativa**.
- K>1, S>1 : **Caché Associativa a K-Camins**.

Les diferents organitzacions de dades a la caché tenen repercussions en:

- Com podrem col·locar la informació (blocs) a la caché.
- Com es fa la cercar de la informació a la caché.
- Les prestacions i complexitat del sistema de caché.

ORGANITZACIÓ AMB MAPEJAT DIRECTE

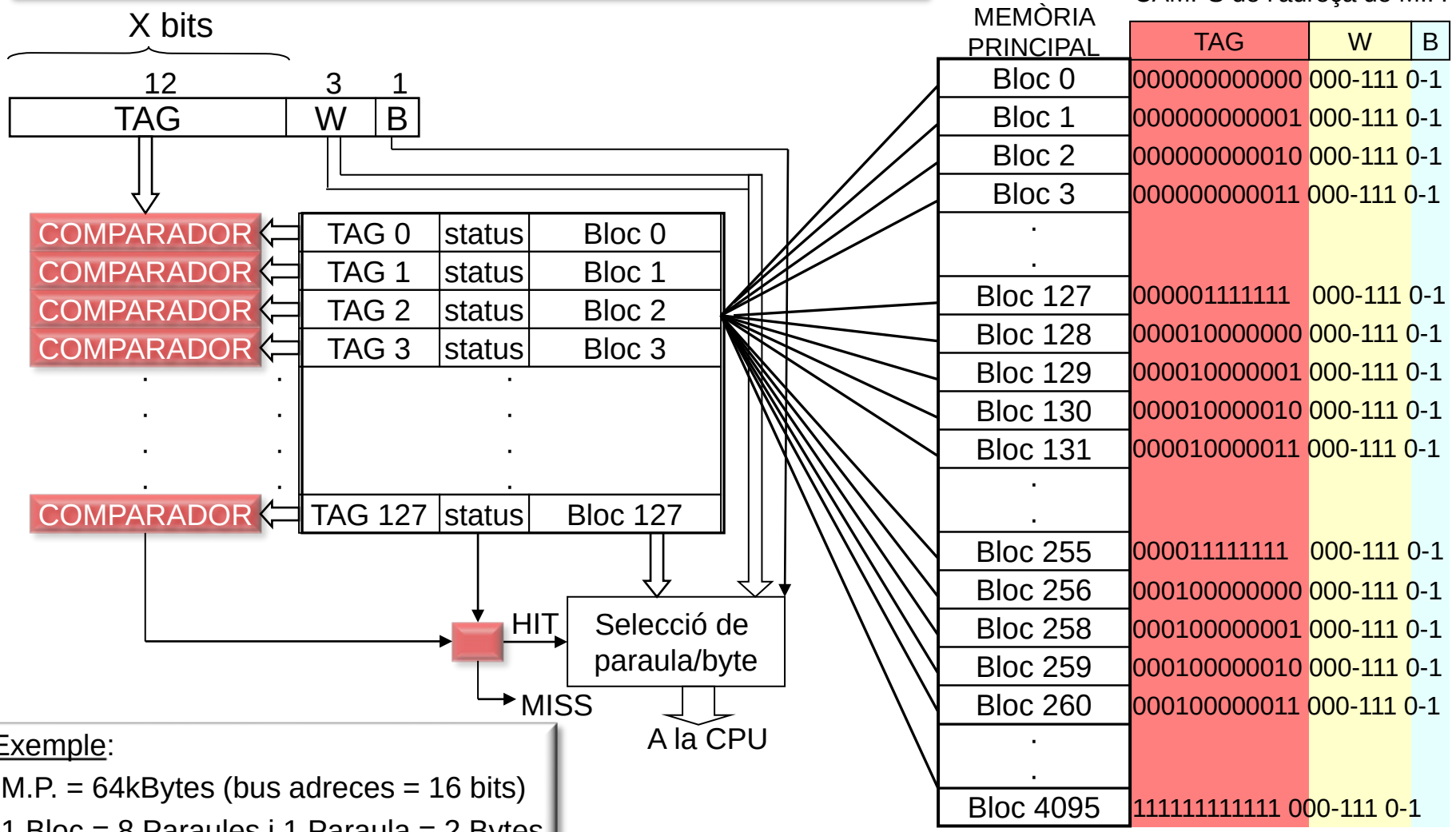


Exemple:

- M.P. = 64kBytes (bus adreces = 16 bits)
- 1 Bloc = 8 Paraules i 1 Paraula = 2 Bytes
  Es a dir M.P.= 4kBlocs
- Caché de 128 Línies (blocs)

Cada bloc de M.P. tant sols pot anar a una línia determinada a la Caché

ORGANITZACIÓ COMPLETAMENT ASSOCIATIVA

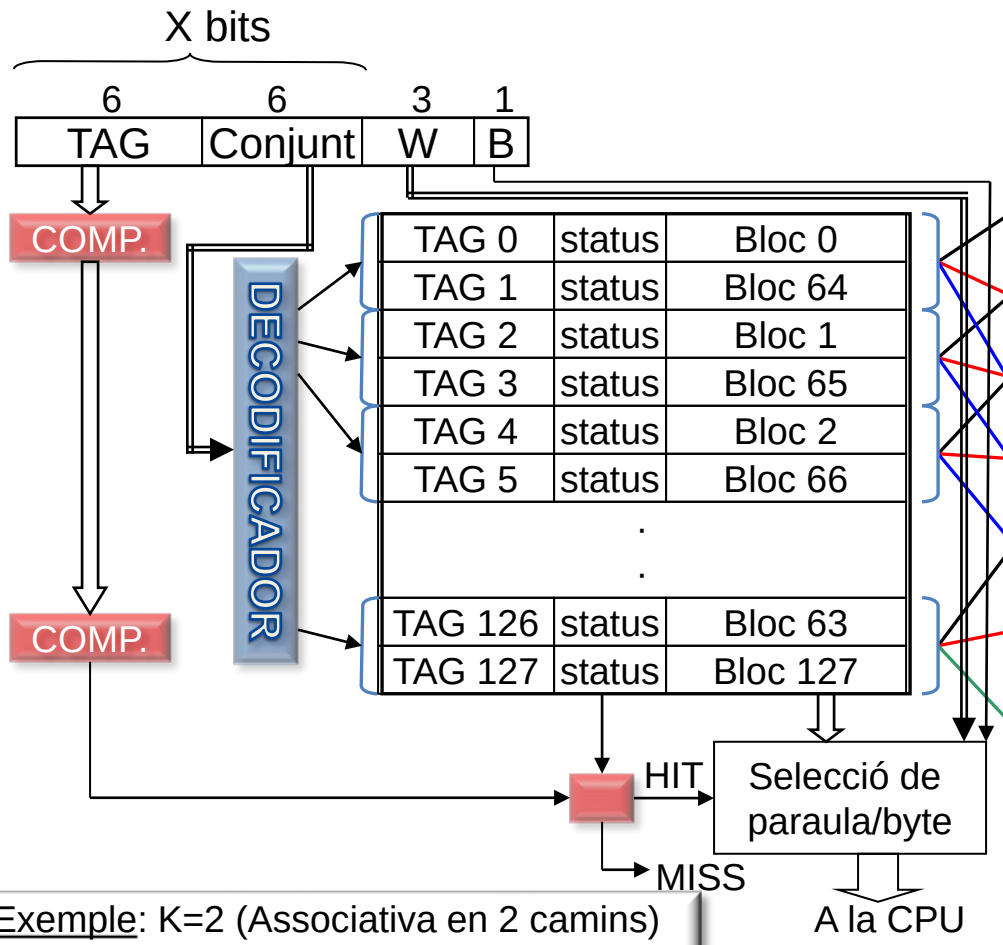


Exemple:

- M.P. = 64kBytes (bus adreces = 16 bits)
- 1 Bloc = 8 Paraules i 1 Paraula = 2 Bytes
  Es a dir M.P.= 4kBlocs
- Caché de 128 Blocs

Qualsevol bloc de M.P. pot estar a qualsevol línia a la Caché.

ORGANITZACIÓ ASSOCIATIVA EN K-CAMINS



Exemple K=2

MEMÒRIA PRINCIPAL	CAMPS de l'adreça de M.P. TAG	Conjunt	W	B
Bloc 0	000000	000000	000-111	0-1
Bloc 1	000000	000001	000-111	0-1
Bloc 2	000000	000010	000-111	0-1
.				
.				
Bloc 63	000000	111111	000-111	0-1
Bloc 64	000001	000000	000-111	0-1
Bloc 65	000001	000001	000-111	0-1
Bloc 66	000001	000010	000-111	0-1
.				
.				
Bloc 127	000001	111111	000-111	0-1
Bloc 128	000010	000000	000-111	0-1
Bloc 129	000010	000001	000-111	0-1
Bloc 130	000010	000010	000-111	0-1
.				
.				
Bloc 4095	111111	111111	000-111	0-1

Exemple: K=2 (Associativa en 2 camins)
- M.P. = 64kBytes (bus adreces = 16 bits)
- 1 Bloc = 8 Paraules i 1 Paraula = 2 Bytes
  Es a dir M.P.= 4kBlocs
- Caché de 128 Blocs

- 2 Camins: Cada bloc pot estar a dos línies diferents a la Caché.
- 6 bits de conjunt: Hi ha 2^6 conjunts (64).

Resum d'Avantatges i Desavantatges de les organitzacions

	Mapejat Directe	Completament Associativa	Associativa K-Camins
EL MILLOR	Lògica Senzilla (hardware més senzill)	Flexibilitat.	Compromís entre les dues anteriors. Quan més gran sigui K, millors prestacions però més complexa. (habitualment K és 2, 4, 8, 16)
EL MILLOR	Millor relació : "n_TX_útils"/n_TX_Totals"	Millors prestacions (hit rate)	
EL MILLOR	No es necessita algoritme de reemplaçament (software més senzill)		
EL PITJOR	Les pitjors prestacions	Complexitat a nivell de hardware.	
EL PITJOR		Només factibles per cachés molt petites.	

n_TX = número de transistors

A més dels camps TAG i Bloc d'informació, a cada línia de caché tenim un camp amb el que denominem bits de status que tenen diverses funcions:

Bit de status	Estat	Significat/Funció	Raó
Line Valid Bit (1 bit per línia)[1]	Activat	Les dades a aquesta línia són vàlides.	
	Desact.	Encara que estan a aquesta línia, les dades NO són vàlides.	Algun altre dispositiu ha modificat les dades a MP i no a aquesta caché.
Dirty/Modified Bit (1 bit per línia)[1]	Activat	Les dades són vàlides però no són iguals a les de MP.	S'ha modificat la informació a caché però no a MP (política escriptura WB)
	Desact.	Les dades són vàlides i a més coincideixen amb les de MP.	
Camp LRU	[2]	Serveix per determinats algoritmes de reemplaçament.	
Coherència	[2]	Per mantenir coherència al fer servir la informació al sistema.	
Drets Accés	[2]	Per saber si un procés té o no dret a la informació de la línia.	
Paritat (1 bit per línia)[1]	[2]	Permet fer un mínim control d'errors.	

Quan hem de portar dades a caché de MP (o d'una caché de nivell superior) i aquesta està plena*, hem de treure algun bloc de la caché per portar el bloc on està la informació de volem.

El problema és quin traiem, perquè si fem fora un bloc que farà falta aviat perdrem prestacions. Aquest punt és molt important per tenir un sistema de caché eficaç.

Algoritmes de Reemplaçament:

- Per a cachés de **Mapejat Directe no fa falta cap algoritme**, ja que un bloc de caché no pot anar només que a una línia de caché (té només un camí).
- **FIFO** (*First In First Out*): El bloc que va arribar dels que hi ha al conjunt serà el primer en sortir. Marxa el que fa més temps que està. Emprat en caches "molt" associatives.
- **LRU** (*Least Recently Used*): El bloc que fa més temps que no fem servir al conjunt és el que substituirem pel nou.
- **MRU** (Most Recently Used): Just al contrari que l'anterior.
- **LFU** (*Least Frequently Used*): Traiem el bloc que hem fet servir menys vegades.
- **RANDOM**: Triem a l'atzar el que marxarà.

* Plena en moltes situacions no vol dir que no hi hagi cap línia buida a la caché, sinó que el conjunt on ha d'estar el bloc que volem portar no té cap línia (camí) buida.

Algoritme de reemplaçament LRU

El nostre objectiu serà mantenir a caché els blocs que es faran servir aviat.

La propietat de “Referències Locals” (els programes estan localitzats a zones properes durant períodes de temps raonables) ens dóna un algoritme:

- Quan s’ha de treure un bloc, triarem el que fa més temps que no fem servir.

Per poder implementar aquest algoritme, hem de anotar “quin va ser l’últim cop que fem servir cada bloc” de cada conjunt. Això ho fem afegint un camp als bits de status de cada línia de caché: camp LRU.

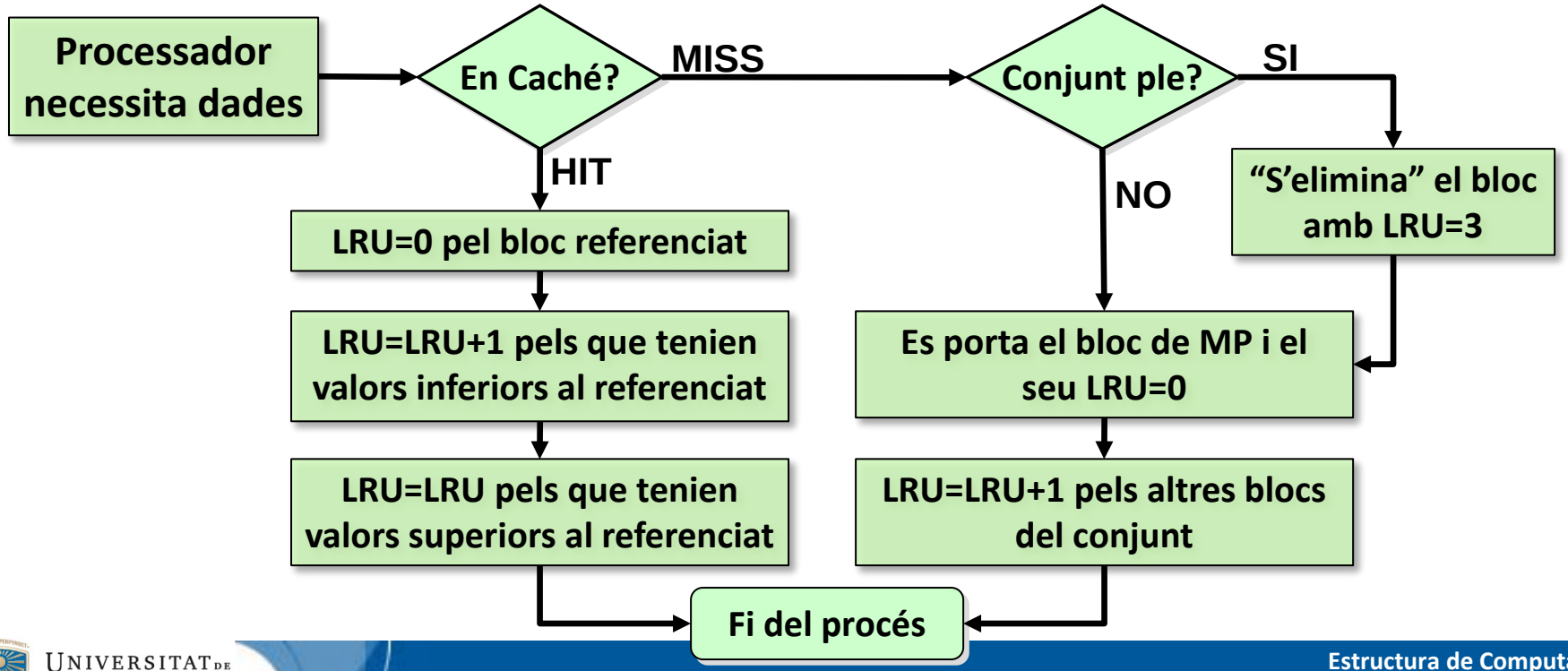
El camp LRU ha de tenir tants bits com el $\log_2$ del nombre de línies (camins) que té el conjunt.

P.ex. Si K=8 necessitarem 3 bits pel camp LRU, si K=4 el camp LRU serà de 2 bits...

Exemple de funcionament de l'algoritme de reemplaçament LRU

Suposem una caché associativa a 4 camins (K=4).

- El camp LRU ha de tenir 2 bits.
- Aquest camp, tindrà un valor entre 0 i 3 segons el temps que fa que no hem fet servir informació del bloc associat respecte als altres blocs del seu conjunt. LRU=3 el que fa més temps, LRU=0 el que fa menys.



Exemple de funcionament de l'algoritme de reemplaçament LRU

Conjunt "i" de caché	
LRU	BLOC

1.- Inici, conjunt està buit

Conjunt "i" de caché	
LRU	BLOC
0	BLOC_A

2.- Necessitem el bloc_A, ve de MP

Conjunt "i" de caché	
LRU	BLOC
1	BLOC_A
0	BLOC_B

3.- Necessitem el bloc_B, ve de MP

Conjunt "i" de caché	
LRU	BLOC
2	BLOC_A
1	BLOC_B
0	BLOC_C

4.- Necessitem el bloc_C, ve de MP

Conjunt "i" de caché	
LRU	BLOC
3	BLOC_A
2	BLOC_B
1	BLOC_C
0	BLOC_D

5.- Necessitem el bloc_D, ve de MP

Conjunt "i" de caché	
LRU	BLOC
3	BLOC_A
2	BLOC_B
0	BLOC_C
1	BLOC_D

6.- Tornem a necessitar el bloc_C

Conjunt "i" de caché	
LRU	BLOC
0	BLOC_A
3	BLOC_B
1	BLOC_C
2	BLOC_D

7.- Tornem a necessitar el bloc_A

Conjunt "i" de caché	
LRU	BLOC
1	BLOC_A
0	BLOC_E
2	BLOC_C
3	BLOC_D

8.- Necessitem el bloc_E, ve de MP

- En vermell/negreta el bloc que fem servir.
- En blau/negreta els camps LRU modificats.
- En blanc sense canvis.