Arquitectura de Computadoras para Ingeniería

(Cód. 7526)
1° Cuatrimestre 2016

Dra. Dana K. Urribarri
DCIC - UNS

Memoria
(continuación)

Tiempo de acceso efectivo

- La *tasa de aciertos* (*hit ratio*) es la fracción (porcentaje) de accesos exitosos.
- La *tasa de fallos* (*miss ratio*) es la fracción (porcentaje) de accesos fallidos.

$$\textit{Miss ratio} = 1 - \textit{hit ratio}$$

Tiempo de acceso efectivo

- El tiempo efectivo de acceso a un nivel L_i será:

$$T_{\text{efectivo Li}} =$$

$$= \textit{hit ratio}_{\text{Li}} * T_{\text{acceso Li}} + \textit{miss ratio}_{Li} * T_{\text{acceso Li+1}}$$

$$= \textit{hit time}_{\text{Li}} + \textit{miss ratio}_{\text{Li}} * \textit{miss penalty}_{\text{Li}}$$

- Hit time: es el tiempo de acceso al nivel L_i para obtener el dato o deducir que hay un faltante.
- Miss penalty: es el tiempo que demanda obtener el bloque en el nivel L_{i+1} y transferirlo al nivel L_i.

Ejemplo

- L_i = cache y L_{i+1} = memoria principal
- Relación 1 a 10 en tiempos de acceso
- Con una tasa de aciertos en cache del 90%

$T_{efectivo\ cache}$ =

= $hit\ ratio_{cache} * T_{acceso\ cache} + miss\ ratio_{cache} * T_{acceso\ MP}$

= $0.9 * 1 + 0.1 * 10$

✓ = 1.9 ⟶ Está más cerca de 1 que de 10.

Ejemplo

- L_i = mem. principal y L_{i+1} = mem. secundaria
- Relación 1 a 10^6 en tiempos de acceso
- Con una tasa de aciertos en MP del 90%

$T_{efectivo\ MP}$ =

= $hit\ ratio_{MP}$ * $T_{acceso\ MP}$ + $miss\ ratio_{MP}$ * $T_{acceso\ MS}$

= $0.9 * 1 + 0.1 * 10^6$

✘ = 100000.9 → Es casi 10^5.
Está más cerca de 10^6 que de 1.
Recién con una tasa de aciertos del 99,99% se logra un tiempo efectivo de 100.

Tiempo de acceso efectivo

- La disparidad en los tiempos de acceso entre el nivel L_i y el nivel L_{i+1} influye fuertemente en el tiempo de acceso efectivo al nivel L_i.
- A mayor disparidad entre niveles consecutivos, mayor es la tasa de aciertos que se requiere.

Tiempo de acceso efectivo

- En la penalización de un faltante (miss penalty) influye
  - tiempo de acceso
  - tiempo de transferencia → depende del tamaño del bloque.
- Mayor tamaño de bloque reduce los faltantes por localidad espacial.
- Mayor cantidad de bloques reduce los faltantes por localidad temporal.
- El tamaño del bloque tiene que ser balanceado.

Memoria cache

Memoria caché

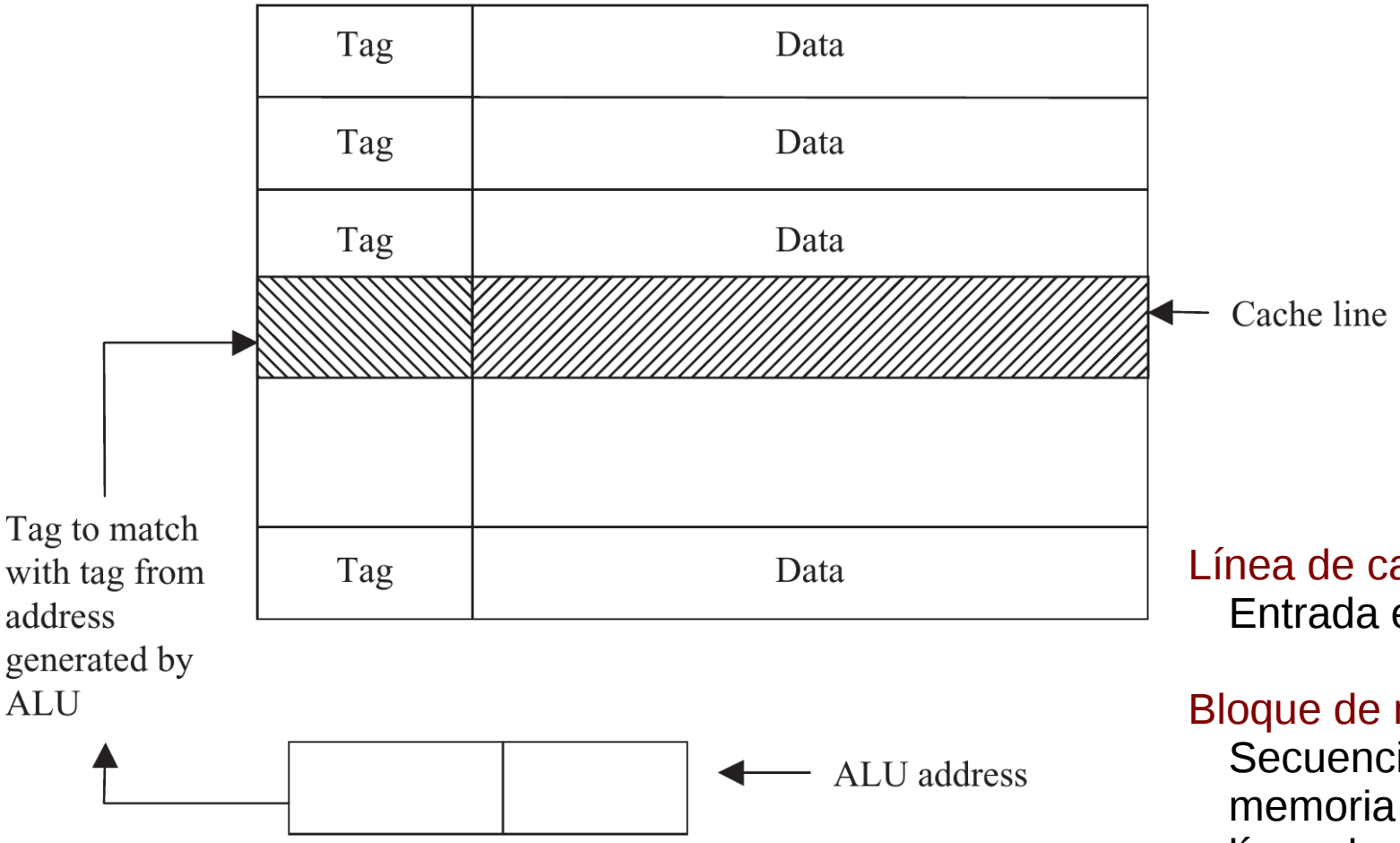
- Sirve como *buffer* de alta velocidad entre memoria principal (MP) y el CPU
- La capacidad de la cache es menor que la de MP.
- 4 preguntas para diseñar una caché:
  1) ¿Cuándo llevar el contenido de MP a caché?
  2) ¿Dónde ponerlo?
  3) ¿Cómo sabemos si está en caché?
  4) ¿Qué pasa si hay que llevar nuevo contenido a caché y ésta está llena (o el lugar donde iría el contenido está ocupado)?

Memoria caché

- En una organización básica de caché:
  1) En principio, el contenido se lleva a la caché bajo demanda (cuando hay un faltante)
  2) La caché se divide en entradas. La organización de la caché determina el mapeo de posiciones de memoria a entradas en la caché.
  3) Cada entrada de la caché contiene el nombre (o tag) y el contenido.

  4) Cuando hay un faltante en la caché y la nueva entrada colisiona con alguna existente, un algoritmo de reemplazo resuelve qué entrada reemplazar.

Organización básica de la caché



Línea de caché
Entrada en la caché

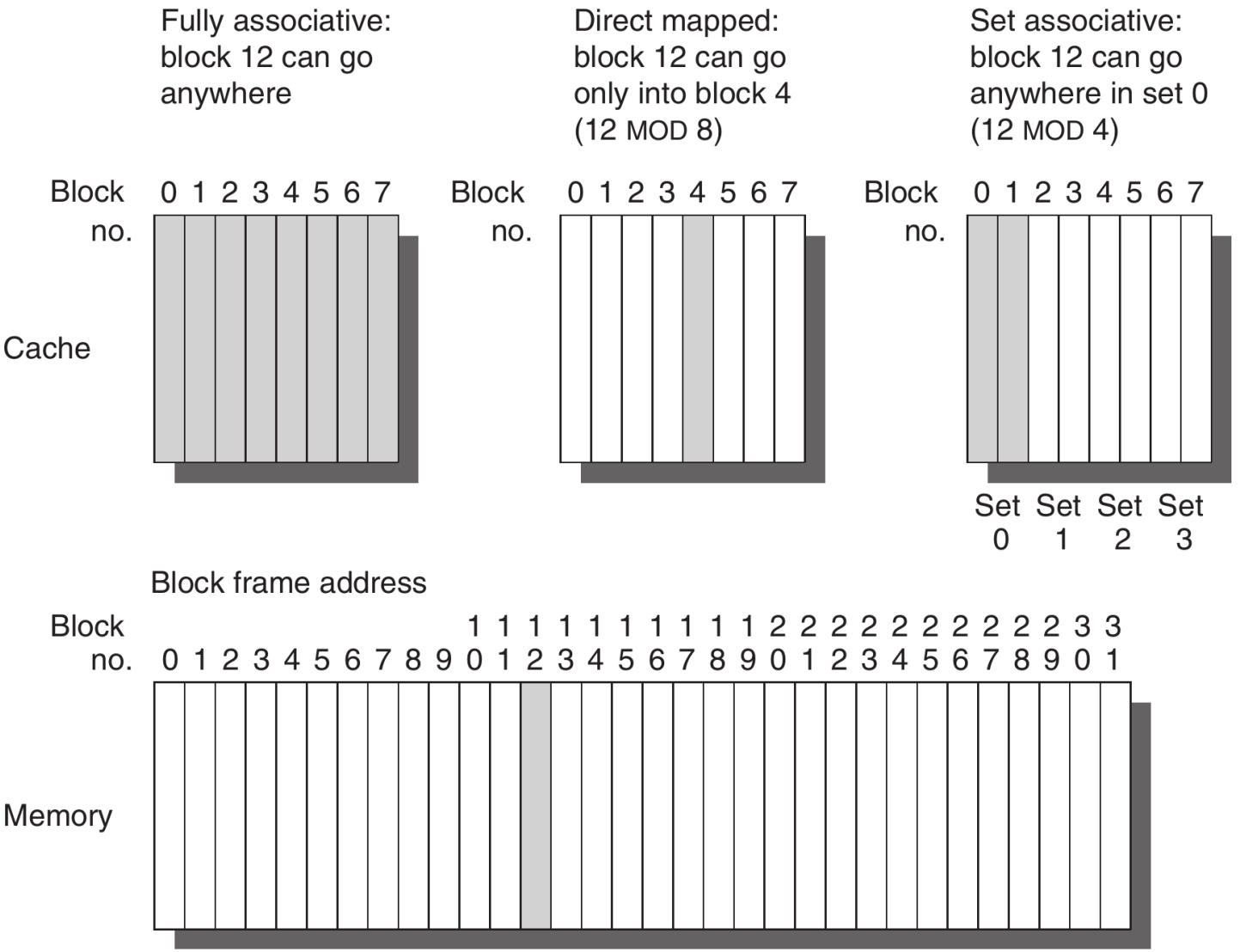
Bloque de memoria:
Secuencia de posiciones de memoria del tamaño de una línea de caché.

Organización de la caché

La organización de la caché determina el mapeo Bloques de Memoria → Líneas de caché:

- Caché *Fully Associative*:
  - Un bloque de memoria puede mapearse a cualquier entrada de la caché.
- Caché *Direct-mapped*:
  - Un bloque de memoria puede mapearse a una única entrada de la caché.
- Caché *n-way Set-Associative*:
  - Un bloque de memoria puede mapearse a un conjunto de *n* entradas de la caché.

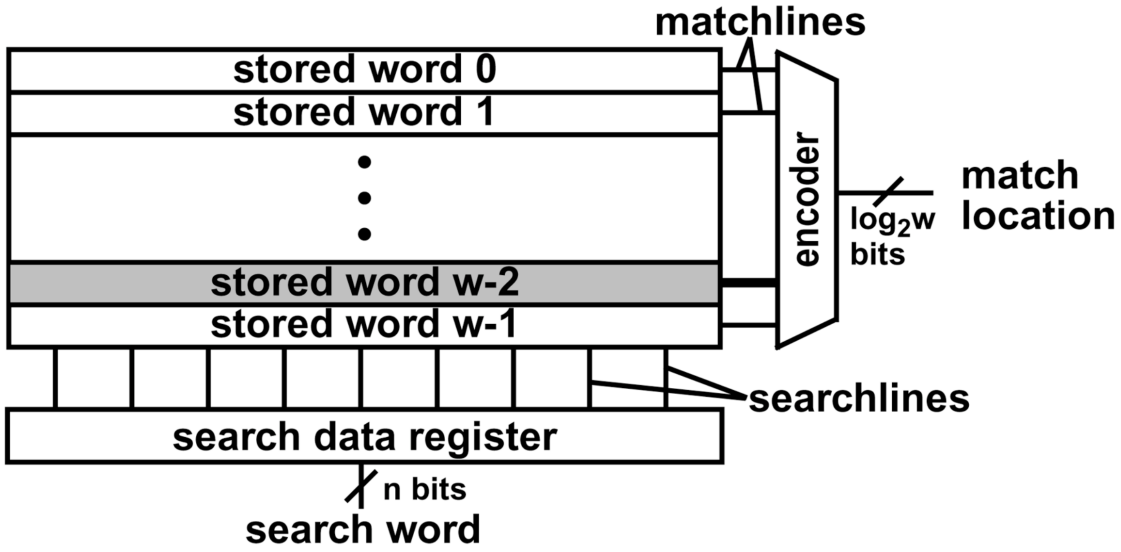
Organización de la caché



Content-addressable memory

- Las caché *fully associative* grandes no son prácticas.
- La búsqueda lineal se puede evitar usando *content-addressable memories* (CAM).
- No se direcciona con un índice.
- Se direcciona en función del contenido.

  Si el contenido existe en la memoria se retorna el índice de la fila correspondiente.



Content-addressable memory

- Desventajas
  1) Mucho más costosas en hw que las SRAM.
  2) Mayor consumo
  3) Difíciles de modificar
- En general son pequeñas.
- No se usan para caché de propósito general.

Direct-mapped caché

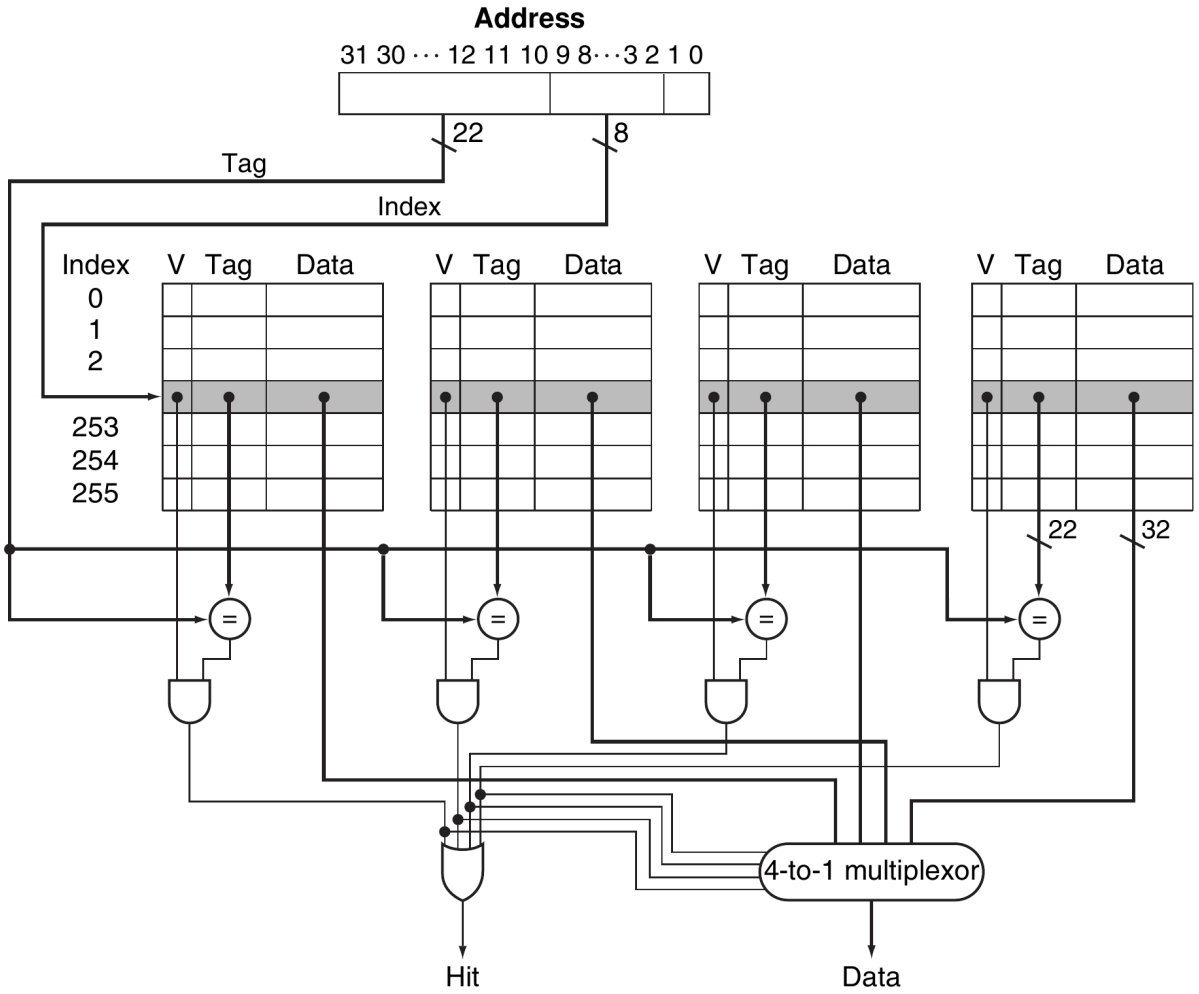
- El bloque de memoria se mapea a una única entrada de la caché.
- Si
  - C (= 2^n) es la cantidad de entradas de la caché
  - A es la dirección de un bloque de memoria B

Al bloque B le corresponde la entrada:

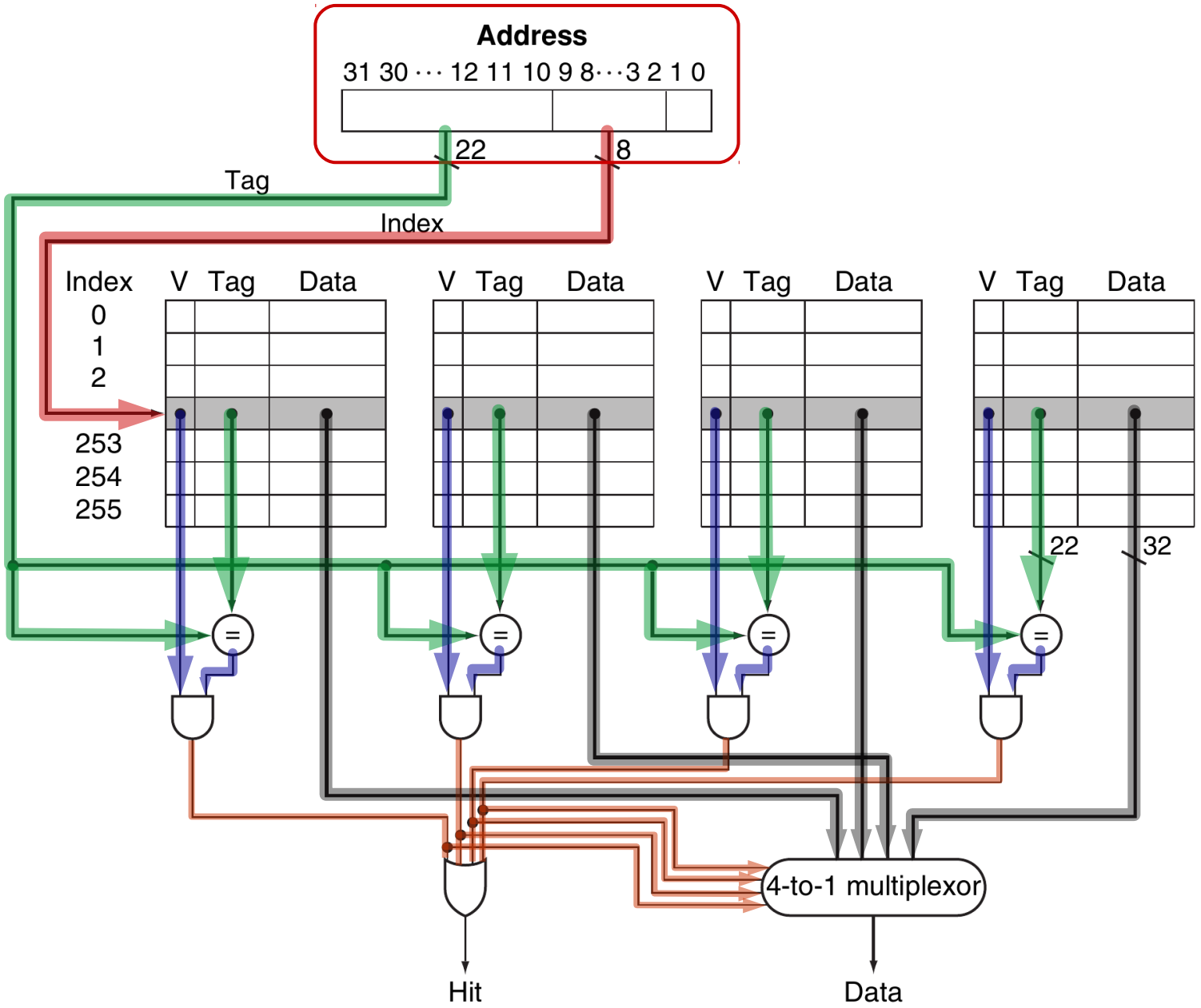
$$A \bmod C$$

M-way set-associative caché

- La memoria se divide en *m* bancos de *C/m* líneas por banco.
- Detectar si hay un bloque en la cache necesita *m* comparadores en paralelo.

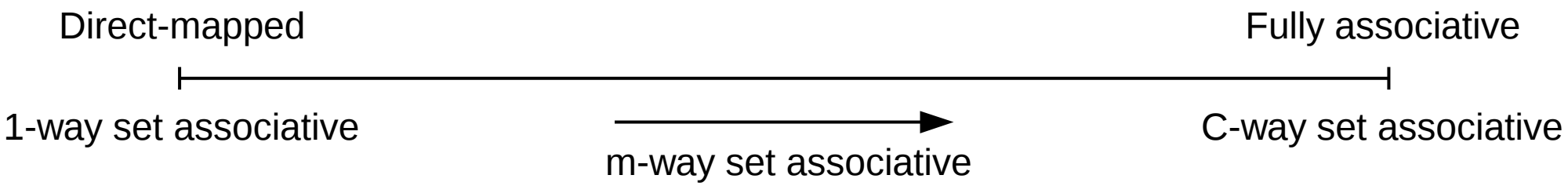


M-way set-associative caché



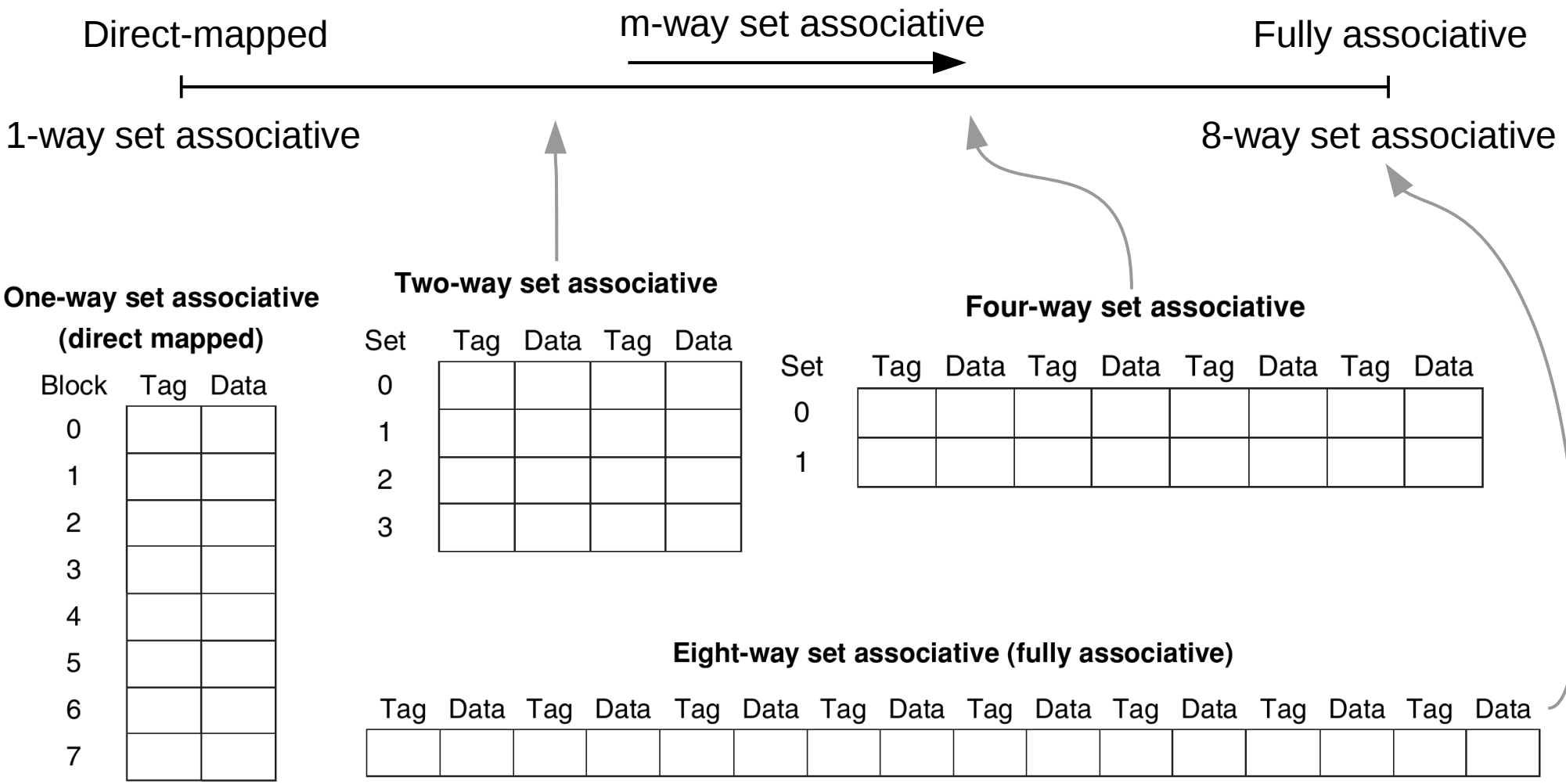
Geometría de la caché

- Parámetros que definen la caché
  - Números de líneas de la caché *C*
  - Tamaño de la línea *L*
  - Grado de asociatividad *m*
- El tamaño de la caché es
  - S = L × C



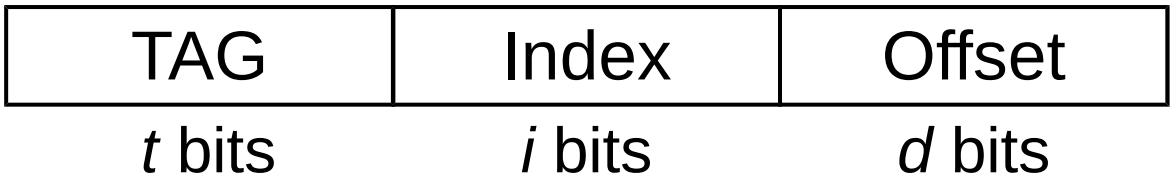
Geometría de la caché

8 líneas de caché



Acceso a la caché

- La ALU genera una dirección de memoria.
- ¿Cómo sabe el hw si la referencia está o no en caché?
- La dirección de memoria se descompone en 3 campos:



  - *Offset*: indica el byte dentro de la línea al cual se está direccionando. Se divide en 2 partes.
  - *Index*: indica la línea de caché (en todos los bancos) en la que hay que revisar si el TAG está o no.

Acceso a la caché

- Si la línea tiene *L* bits, el offset debe tener *d* bits

$$d = \text{Log}_2 L$$

- Si hay C líneas en total, divididas en m bancos, cada banco tiene C/m líneas por lo tanto para el index se necesitan i bits

$$i = \text{Log}_2 C/m$$

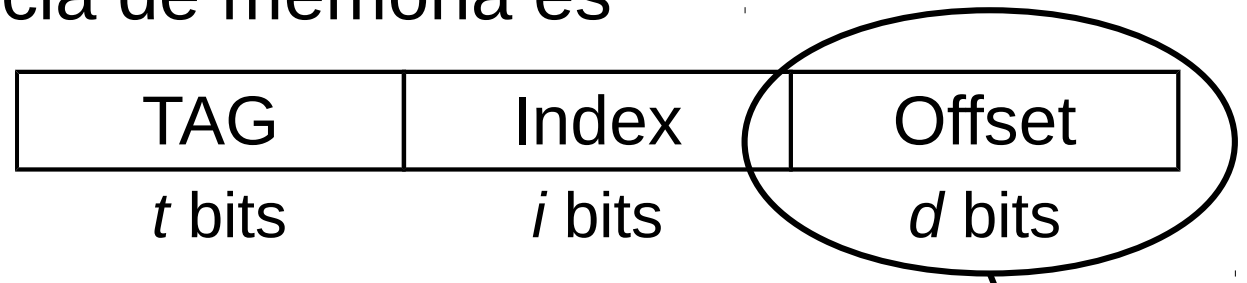
- Los bits restantes son los *t* bits del TAG

Ejemplo

- Direcciones de 32 bits
- Geometría de la caché (*S*,*m*,*L*) = (32KB, 1, 16B)
- El número de líneas *C* en cada banco es

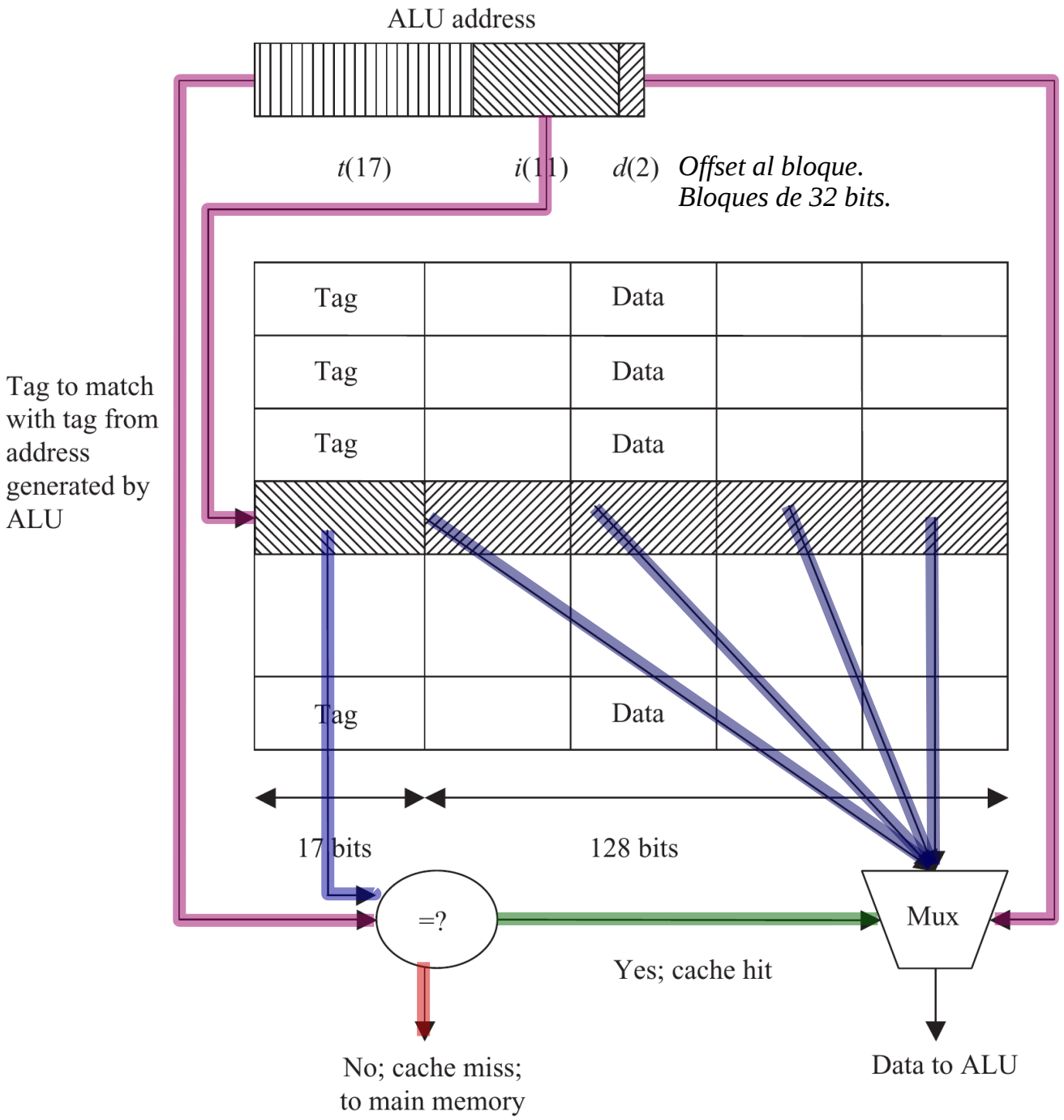
$$32 \times 1024 / 16 = 2048$$

- La referencia de memoria es



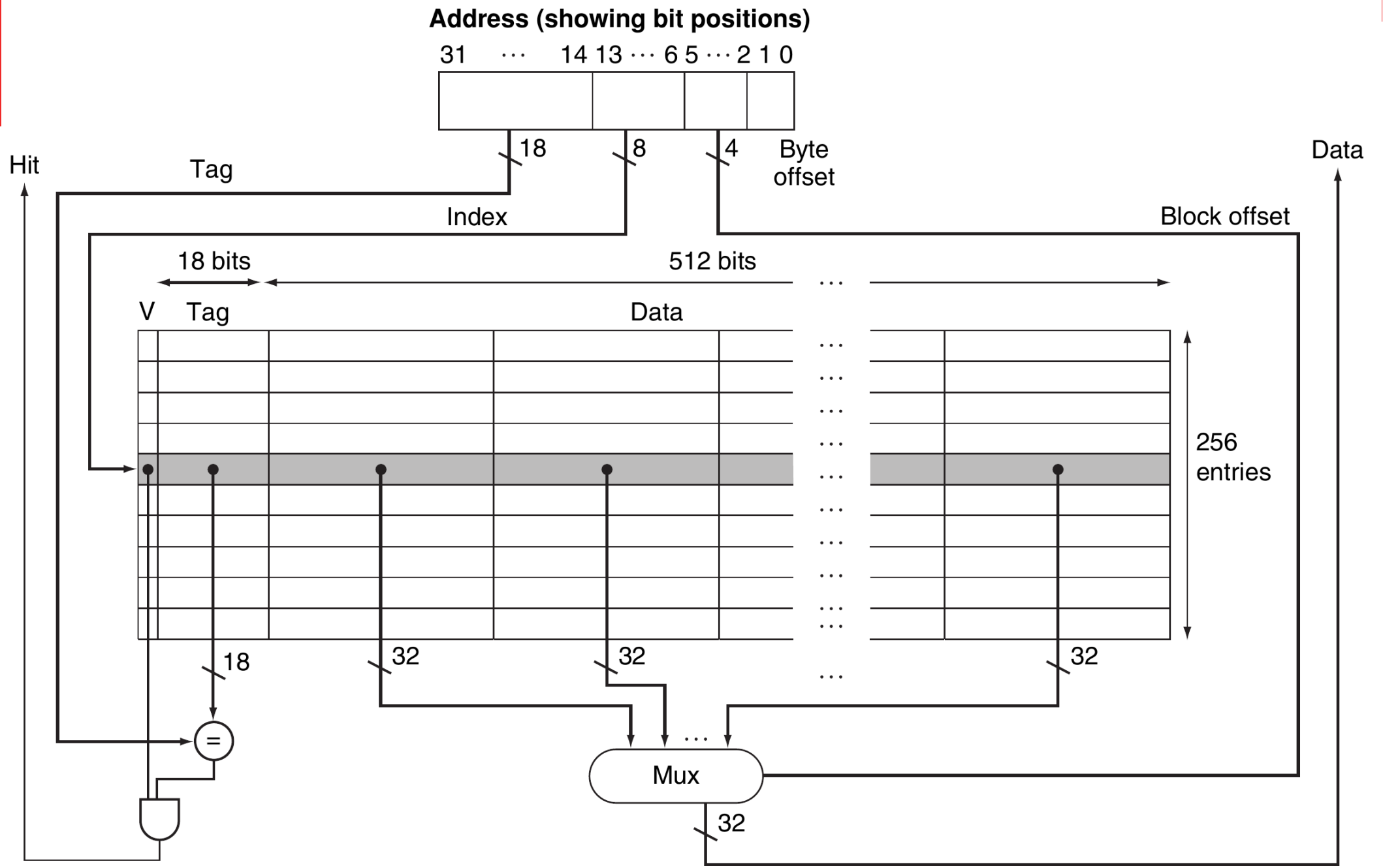
- Donde
  - $d = \text{Log}_2 L = \text{Log}_2 16 = 4$
  - $i = \text{Log}_2 C/m = \text{Log}_2 2048 = 11$
  - $t = 32 - 11 - 4 = 17$

Se divide en offset de la línea y offset del byte dentro de la palabra.


ALU address
t(17)
i(11)
d(2)
Offset al bloque.
Bloques de 32 bits.
Tag
Data
Tag to match with tag from address generated by ALU
17 bits
128 bits
=?
Mux
Yes; cache hit
No; cache miss; to main memory
Data to ALU

Cambios en la geometría

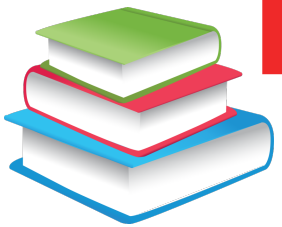
S	m	L	C	d	i	t
			S/m	Log_2 L	Log_2 C/m	32 – d – i
32KB	1	16B	2048	4	11	17
32KB	1	32B	32×1024 / 32 = 1024	Log_2 32 = 5	Log_2 1048 = 10	32–10–5 = 17
32KB	2	16B	32×1024 / 16 = 2048	Log_2 16 = 4	Log_2 2048/2 = 10	32–10–4 = 18
32KB	2048	16B	32×1024 / 16 = 2048	Log_2 16 = 4	Log_2 2048/2048 = 0	32 – 0 – 4 = 28


Address (showing bit positions)
31 ··· 14 13 ··· 6 5 ··· 2 1 0
18
8
4
Byte offset
Hit
Tag
Index
Block offset
Data
18 bits
512 bits
V
Tag
Data
256 entries
18
32
32
32
=
Mux
32

Tamaño de la caché

- El tamaño real de la caché incluye:
  - Bits de tag
  - Bits adicionales (bit de válido)
  - Datos
- En general, solo se hace referencia a la capacidad para datos (S)

Bibliografía



- *Capítulo 5 y Apéndice B.* David A. Patterson & John L. Hennessy. *Computer Organization and Design. The Hardware/Software Interface.* Elsevier Inc. 2014, 5ta Ed.
- *Capítulo 2*. *Multiprocessor Architecture. From simple pipelines to chip multiprocessor*. Jean-Loup Baer. Cambridge University Press. 2010.

Suplementaria

- *Apéndice B.* John L. Hennessy & David A. Patterson. *Computer Architecture. A Quantitative Approach*. Elsevier Inc. 2012, 5ta Ed.