Esercitazione su Gerarchie di Memoria

Introduzione

Memoria

- *gerarchie di memoria*: cache, memoria principale, memoria di massa etc. (con possibilità di fallimenti nell'accesso)
- organizzazione, dimensionamento, indirizzamento, gestione dei conflitti
- influenza sulle prestazioni della macchina

Gerarchie di Memoria

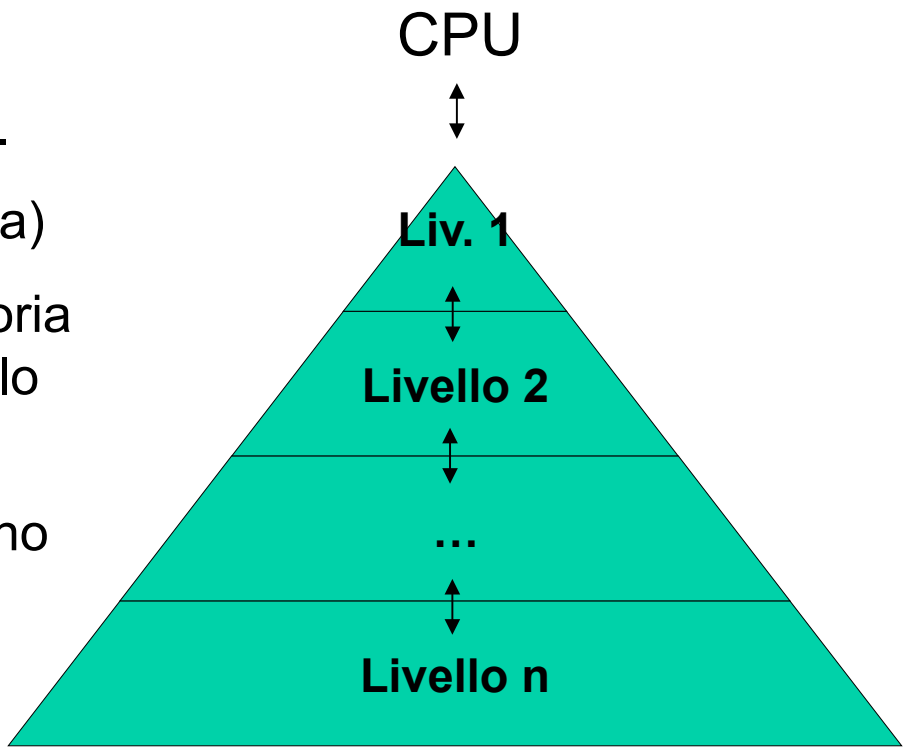
La memoria e organizzata in modo *gerarchico* (a livelli):

Liv. 1: memoria più veloce(costosa).

Liv. n: memoria più lenta (economica)

Il processore indirizza tutta la memoria di *Liv.n*, ma accede direttamente solo al *Liv. 1*.

Per *località* i blocchi cercati si trovano con alta probabilità a livello alto.



Obiettivo: dare l'illusione di avere un memoria veloce come a *Liv. 1* e grande come a *Liv. n*.

Prestazioni e memoria: Esercizio 1

Si consideri l'esecuzione di un programma *P* su di una data CPU. Il *CPI* ideale è pari a *3*, ma considerando i *miss* della cache si ottiene un *CPI* reale pari a *3.6*. Sapendo inoltre che *Miss penalty* = 12 cicli e che *Instruction miss rate* = *4%* determinare *Data miss rate* per il programma considerato, tenendo conto che la percentuale di load/store è del 40%.

$$\begin{aligned} \textit{cicli tot.} &= CPI_{reale} \bullet IC \\ &= CPI_{ideale} \bullet IC + \\ &\quad (IC \bullet \textit{perc. load/store} \bullet \textit{Data miss rate} \bullet \textit{Miss penalty}) + \\ &\quad (IC \bullet \textit{Instruction miss rate} \bullet \textit{Miss penalty}) \end{aligned}$$

$$\begin{aligned} CPI_{reale} &= \textit{cicli tot.} / IC = \\ &= CPI_{ideale} + \textit{perc. load/store} \bullet \textit{Data miss rate} \bullet \textit{Miss penalty} + \\ &\quad + \textit{Instruction miss rate} \bullet \textit{Miss penalty} = \\ &= 3 + 0.4 \bullet \textit{Data miss rate} \bullet 12 + 0.04 \bullet 12 \\ &= 3 + 4.8 \bullet \textit{Data miss rate} + 0.48 \end{aligned}$$

Data miss rate $= 3.6 - 3.48 / 4.8 = 0.025$ cioè *2.5%*

Gerarchie di memoria: Cache

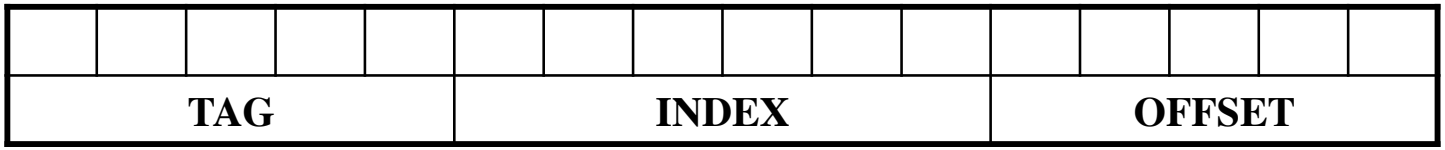
Se l'indirizzo fisico è di *m bit* e la dimensione del blocco è 2^n, allora gli *m - n bit* più significativi rappresentano l'indirizzo del blocco, i rimanenti l'offset all'interno del blocco.

- Cache diretta

Ogni blocco di memoria può essere inserito in un'unica entry della cache.

- Cache n-way associative

I blocchi della cache sono suddivisi in insiemi di dimensione *n*. Ogni blocco di memoria può essere inserito in uno degli *n* blocchi di un insieme (riduce la possibilità di conflitti).



Gerarchie di memoria: Cache

Consideriamo ora i problemi relativi al dimensionamento, organizzazione, indirizzamento della memoria cache …

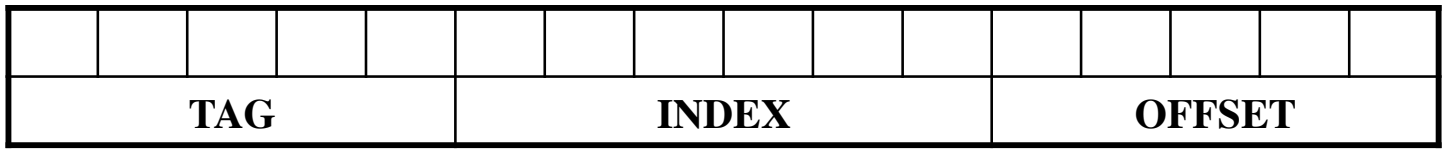
Se l'indirizzo (memoria principale) è di *m bit* e la dimensione del blocco è 2^n, allora gli *m - n bit* più significativi rappresentano l'indirizzo del blocco, i rimanenti l'offset all'interno del blocco.

Cache diretta

Ogni blocco di memoria può essere inserito in un unico blocco della cache

Cache block index = Indirizzo del blocco **mod** *#(cache blocks)*

Se *#(cache blocks)* = 2^k allora il *Cache block index è* dato dai *k bit* meno significativi del *Mem. block address.*



Gerarchie di memoria: Cache

Cache n-way associative

I blocchi della cache sono suddivisi in *set* (insiemi) di dimensione *n*. Ogni blocco di memoria può essere inserito in uno degli *n* blocchi di un set (riduce la possibilità di conflitti).

Il numero dei set di blocchi in cache è

$$\#(\text{cache sets}) = \#(\text{cache blocks}) / n$$

Mentre, per un dato blocco di memoria, l'indice del set nella cache sarà:

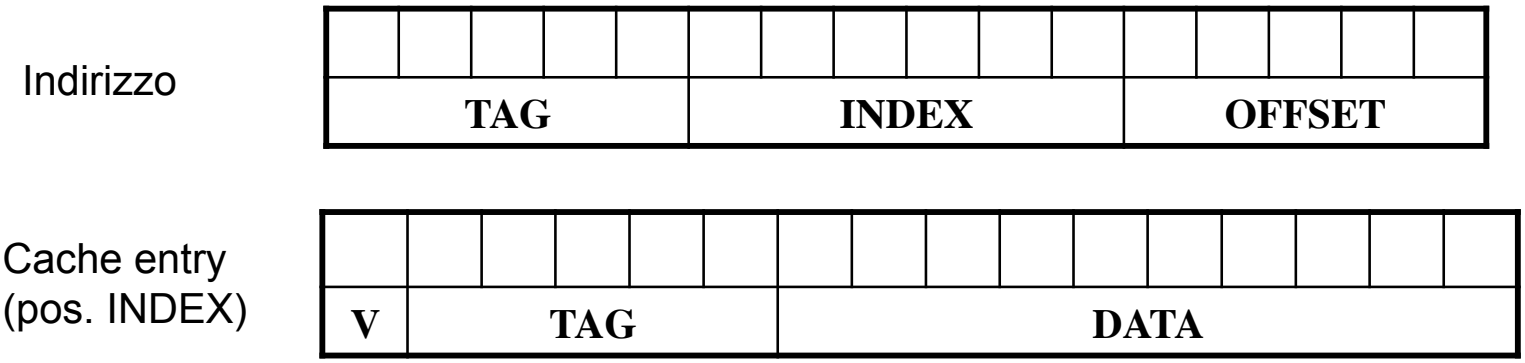
$$\text{Cache set index} = \text{Mem. block address} \ \textbf{mod} \ \#(\text{cache sets})$$

Se $\#(\text{cache sets}) = 2^k$ allora il *Cache set index è* dato dai *k bit* meno significativi del *Mem. block address.*

Gerarchie di memoria: Cache

Come si fa a sapere se il contenuto della cache è il dato cercato?

Le entry della cache contengono le informazioni necessarie:



Si reperisce la cache entry di indirizzo INDEX. Se questa è valida e il suo TAG coincide con quello dell'indirizzo → hit, il dato è corretto. Altrimenti si è verificato un miss.

Per cache *associative* occorre confrontare il TAG dell'indirizzo con quello di *ogni* entry del *set* indirizzato da INDEX.

Cache: Esercizio 2

Considerare una cache *4-way associative*, con parte dati di *8 KB* organizzata in blocchi da *32 B*. L'indirizzo fisico è di *16 bit*.

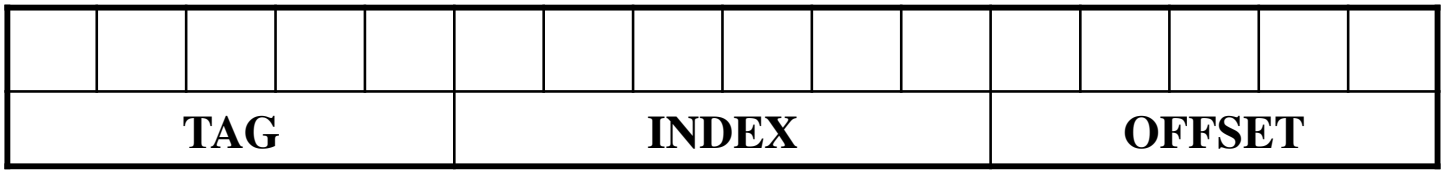
a. Determinare la suddivisione dell'indirizzo fisico nei campi TAG, INDEX, OFFSET

1 Blocco = 32 B $\rightarrow$ *OFFSET* = $log_2 32 = 5$ *bit*

Tot. blocchi = Cache size / Block size = 8KB / 32B = $2^{13} / 2^5 = 2^8$ *blocchi*

Poiché la cache è 4-way associative si hanno $2^8/2^2 = 2^6$ set

Quindi *INDEX = 6 bit* e i rimanenti *5 bit* sono per *TAG*



Cache: Esercizio 2 (continua)

Considerare una cache *4-way associative* in cui la parte dati è di *8 KB* e avente blocchi da *32 B*. L'indirizzo fisico è di *16 bit*.

b. Stabilire se gli indirizzi 0xAFAF e 0xAFB0 sono mappati nello stesso insieme della cache.

			TAG	INDEX	OFFSET
0xAFAF	= 1010 1111 1010 1111	=	10101	111101	01111
0xAFB0	= 1010 1111 1011 0000	=	10101	111101	10000

Quindi i contenuti dei due indirizzi compaiono nello stesso set

c. Supponendo che il contenuto di 0xAFAF sia nella cache, cosa accade se tentiamo di leggere il contenuto di 0xAFB0 ?

I due indirizzi hanno identici INDEX e TAG, quindi sono nello stesso blocco !!
Pertanto se il contenuto di 0xAFAF si trova nella cache, vi sarà anche quello di 0xAFB0.

Cache: Esercizio 2 (continua)

Considerare una cache *4-way associative* in cui la parte dati è di *8 KB* e avente blocchi da *32 B*. L'indirizzo fisico è di *16 bit*.

d. Cosa succede invece per gli indirizzi 0xAFAF e 0xA7B0 ?

						TAG	INDEX	OFFSET
0xAFAF	=	1010	1111	1010	1111	=10101	111101	01111
0xA7B0	=	1010	0111	1011	0000	=10100	111101	10000

Anche in questo caso, i contenuti dei due indirizzi si trovano nello stesso set.

Ma ora i due indirizzi hanno TAG diverso!!

Quindi, se il contenuto di 0xAFAF è nella cache, non è detto che ci sia anche quello di 0xAFB0 [anzi nel caso di cache a corrispondenza diretta (direct mapped) uno escluderebbe l'altro].

Cache: Esercizio 2 (continua)

Considerare una cache *4-way associative* in cui la parte dati è di *8 KB* e avente blocchi da *32 B*. L'indirizzo fisico è di *16 bit*.

					TAG	INDEX	OFFSET
0xAFAF	=	1010	1111	1010 1111	=10101	111101	01111
0xA7B0	=	1010	0111	1011 0000	=10100	111101	10000

e. ... se la cache fosse *2-way associative*?

Tot. blocchi = 2^8

Dato che la cache è 2-way associative si hanno $2^8 / 2 = 2^7$ set

quindi INDEX = 7 bit ... e i due indirizzi considerati sono su set diversi!

					TAG	INDEX	OFFSET
0xAFAF	=	1010	1111	1010 1111	= 1010	1111101	01111
0xA7B0	=	1010	0111	1011 0000	= 1010	0111101	10000

Cache: Esercizio 2 (continua)

Considerare una cache *4-way associative* in cui la parte dati è di *8 KB* e avente blocchi da *32 B*. L'indirizzo fisico è di *16 bit*.

f. Come sono ripartiti i campi TAG, INDEX e OFFSET se invece la cache fosse *completamente associativa*?

1 Blocco = 32 B $\rightarrow$ *OFFSET* = $log_2 32 = 5$ *bit*

Tot. blocchi = Cache size / Block size = 8KB / 32B = $2^{13}/2^5 = 2^8$ *blocchi*

Poichè la cache è completamente associativa vi è un unico set con 2^8 blocchi. Pertanto *INDEX = 0 bit* e *TAG = 16 - 5 = 11 bit*

Cache e conflitti

Memoria cache: qualcosa sui conflitti...

Cache e conflitti: Esercizio 3

Si consideri una cache *1-way associative* (diretta) e si assuma che l'indirizzo fisico sia di *16 bit*, suddiviso in *4 bit* di OFFSET, *4 bit* di INDEX e *8 bit* di TAG.
Si consideri inoltre una sequenza di accessi in lettura alla memoria, agli indirizzi seguenti:

1. `0x7F5A`
2. `0x3CC7`
3. `0x7F5B`
4. `0x10C2`
5. `0x8F50`

Si supponga che inizialmente ogni blocco nella cache sia *non valido*.

a. Quali saranno i blocchi *validi* alla fine della sequenza di letture?

I blocchi di cache interessati dalle operazioni di lettura sono quelli con INDEX = 5 (cioè il sesto blocco) e INDEX = C (cioè il tredicesimo blocco). Questi saranno quindi gli unici blocchi *validi*.

Cache e conflitti : Esercizio 3 (continua)

Si consideri una cache *1-way associative* (diretta) e si assuma che l'indirizzo fisico sia di *16 bit*, suddiviso in *4 bit* di OFFSET, *4 bit* di INDEX e *8 bit* di TAG. Si consideri inoltre la sequenza di accessi in lettura:

1. `0x7F5A`
2. `0x3CC7`
3. `0x7F5B`
4. `0x10C2`
5. `0x8F50`

b. Si verificano conflitti durante la sequenza di lettura? Se sì, quali?

Gli accessi 1 e 3 fanno riferiscono lo stesso blocco di cache, ma anche di memoria sottostante (stesso TAG), quindi non vi è alcun conflitto.

L'accesso 4 è in conflitto con 2 (stesso INDEX, TAG diversi). Quindi 4 rimuoverà dalla cache il blocco corrispondente a 2.

L'accesso 5 è in conflitto con il 3 e 1, quindi rimuoverà dalla cache il blocco corrispondente agli accessi 3 e 1.

Cache e conflitti : Esercizio 3 (continua)

Si consideri una cache *1-way associative* (diretta) e si assuma che l'indirizzo fisico sia di *16 bit*, suddiviso in *4 bit* di OFFSET, *4 bit* di INDEX e *8 bit* di TAG. Si consideri inoltre la sequenza di accessi in lettura:

1. `0x7F5A`
2. `0x3CC7`
3. `0x7F5B`
4. `0x10C2`
5. `0x8F50`

c. Come si presenta la cache (blocchi *validi*) alla fine della sequenza di lettura?

	VALID	TAG	DATA
INDEX: 5	1	8F	Blocco all'indirizzo specificato in 5
INDEX: C	1	10	Blocco all'indirizzo specificato in 4

Cache e conflitti : Esercizio 3 (continua)

c. Si consideri ancora la sequenza di accessi in lettura:

1. `0x7F5A`
2. `0x3CC7`
3. `0x7F5B`
4. `0x10C2`
5. `0x8F50`

ma si supponga che, con la stessa composizione dell'indirizzo (*4 bit* di OFFSET, *4 bit* di INDEX e *8 bit* di TAG), la cache sia *2-way associative*. Si verificano conflitti durante la sequenza di lettura? Se si, quali?

Questa volta INDEX si riferisce ai set, e ogni set contiene due blocchi (quindi in totale i blocchi sono 32).

Visto che nella sequenza ogni set è riferito al più due volte, *non* si verificheranno conflitti.

Cache e conflitti : Esercizio 3 (continua)

Sequenza di letture (*4 bit* di OFFSET, *4 bit* di INDEX e *8 bit* di TAG), …

1. `0x7F5A`
2. `0x3CC7`
3. `0x7F5B`
4. `0x10C2`
5. `0x8F50`

La situazione dei blocchi validi alla fine della sequenza di lettura è:

Set con INDEX = 5

Blocco	VALID	TAG	DATA
0	1	7F	Blocco all'indirizzo specificato in 1 e 3
1	1	8F	Blocco all'indirizzo specificato in 5

Set con INDEX = C

Blocco	VALID	TAG	DATA
0	1	3C	Blocco all'indirizzo specificato in 2
1	1	10	Blocco all'indirizzo specificato in 4

Cache e conflitti: Esercizio 3 (continua)

d. Si consideri ancora la sequenza di accessi in lettura:

1. `0x7F5A` → `0111 1111 0 101 1010`
2. `0x3CC7` → `0011 1100 1 100 0111`
3. `0x7F5B` → `0111 1111 0 101 1011`
4. `0x10C2` → `0001 0000 1 100 0010`
5. `0x8F50` → `0100 1111 0 101 0000`

e si assuma di avere una cache *2-way associative*, ma con lo stesso spazio dati della prima (2^4 blocchi di 2^4 *B*). Si verificano conflitti durante la sequenza di lettura?

I 2^4 blocchi saranno suddivisi in 2^3 set, quindi indirizzati da un INDEX di *3 bit*. OFFSET resta di *4 bit* ed i rimanenti *9 bit* formano il TAG.

Ancora una volta nella sequenza ogni set è riferito al più due volte, quindi *non* si verificheranno conflitti

1. e 3. fanno riferimento allo stesso blocco

Memoria Virtuale

La memoria principale può a sua volta agire come “cache” per la memoria di massa (dischi, nastri).

Si parla in questo caso di memoria virtuale:

- I programmi sono compilati rispetto ad uno *spazio di indirizzamento virtuale*, diverso da quello fisico;
- Il sistema di memoria virtuale *traduce* gli indirizzi virtuali in indirizzi fisici.

Vantaggi:
- illusione di avere una maggiore memoria fisica;
- *rilocazione* dei codici;
- meccanismi di *protezione* (gli spazi virtuali di programmi diversi mappati su spazi fisici differenti).

Memoria Virtuale

I concetti fondamentali relativi alla memoria virtuale sono analoghi a quelli visti per la cache, ma per ragioni storiche la terminologia è diversa.

- Memoria fisica / memoria virtuale

- Pagine (blocchi)

- Page fault (miss) [la pagina non è in memoria e deve essere letta da disco]

Il costo dei *page fault* è estremamente elevato (milioni di cicli), e questo influenza molte scelte (dimensioni della pagina elevate *4, 16, 32, 64 KB*, completa associatività, gestione software, write-back).

Memoria Virtuale: Traduzione degli indirizzi

Ad ogni istante solo una parte delle pagine di memoria “virtuali” è presente nella memoria fisica. L’indirizzo virtuale si suddivide in

VIRTUAL PAGE ADDRESS	OFFSET

Il *Virtual Page Address* indirizza le entry di una tabella, detta *Page Table*, che contiene informazioni sulla presenza / validità delle pagine in memoria

V	D	PHYSICAL PAGE NUMBER

Per ragioni di efficienza non si accede direttamente alla *Page Table*, ma piuttosto al *TLB (translation Lookaside Buffer)* che svolge il ruolo di “cache” della *Page Table* …

Memoria Virtuale: Esercizio 1

Sia dato un sistema di memoria virtuale paginata con pagine di *2 KB* e una cache con blocchi di dati di *16 B*. Considerare il seguente loop che accede un array di interi (ognuno memorizzato su *4 B*)

```
for (i=0; i<2048; i++)
  if (a[i]%2)
     a[i]--;
```

Si supponga inoltre che la porzione di memoria virtuale acceduta dal loop non sia inizialmente presente, né in memoria fisica né in cache.

a. Che tipo di località sfrutta il loop?

Spaziale elementi di memoria vicini vengono riferiti in sequenza (istruzioni ed elementi dell'array);

Temporale la stessa locazione viene acceduta sia in lettura che in scrittura.

Memoria Virtuale: Esercizio 1 (continua)

Sia dato un sistema di memoria virtuale paginata con pagine di *2 KB* e una cache con blocchi di dati di *16 B*. Considerare il seguente loop che accede ad un array di interi (ognuno memorizzato su *4 B*)

```
for (i=0; i<2048; i++)
   if    (a[i]%2)
         a[i]--;
```

Si supponga inoltre che la porzione di memoria virtuale acceduta dal loop non sia inizialmente presente, né in memoria fisica né in cache.

b. Indicare il numero di page fault (certi), sia quando l'indirizzo virtuale *&a[0]* è un multiplo della dimensione della pagina, sia quando non lo è.

2048 valori interi → *2048* • *4B* = *8 KB* richiesti per memorizzare l'array.

Se &a[0] è un multiplo della dimensione della pagina, si avranno *8 KB / 2 KB* = *4* page fault. Altrimenti si avranno *5* page fault.

Memoria Virtuale: Esercizio 1 (continua)

Sia dato un sistema di memoria virtuale paginata, con pagine di *2 KB* e una cache con blocchi di dati di *16 B*. Considerare il seguente loop che accede ad un array di interi (ognuno memorizzato su *4 B*)

```
for (i=0; i<2048; i++)
   if    (a[i]%2)
         a[i]--;
```

Si supponga inoltre che la porzione di memoria virtuale acceduta dal loop non sia inizialmente presente, né in memoria fisica né in cache.

c. Indicare il numero di cache miss (certi), sia quando l'indirizzo fisico corrispondente a *&a[0]* è un multiplo della dimensione del blocco di cache, sia quando non lo è.

Se l'indirizzo corrispondente a *&a[0]* è multiplo della dimensione del blocco si avranno $8\ KB / 16\ B = 2^{13}\ B / 2^4\ B = 2^9 = 512$ cache miss.

Altrimenti si avranno *513* cache miss.

Memoria Virtuale: Esercizio 1 (continua)

Sia dato un sistema di memoria virtuale paginata, con pagine di *2 KB* e una cache con blocchi di dati di *16 B*. Considerare il seguente loop che accede ad un array di interi (ognuno memorizzato su *4 B*)

```
for (i=0; i<2048; i++)
   if    (a[i]%2)
         a[i]--;
```

Si supponga inoltre che la porzione di memoria virtuale acceduta dal loop non sia inizialmente presente, né in memoria fisica né in cache.

d. Perché non è stato necessario dare informazioni sull'organizzazione della cache, o sulla politica di rimpiazzamento delle pagine della memoria virtuale?

Perché abbiamo considerato solo *Compulsory Miss*, che dipendono esclusivamente dalle dimensioni (dell'array, blocchi e pagine) e non dalle specifiche organizzazioni dei livelli di memoria.

Memoria Virtuale: Esercizio 2

Si consideri un sistema di memoria virtuale, la cui *Page Table* (con *valid* e *dirty* bit) ha dimensione *8MB.* La dimensione della pagine è *1024 B* e l'indirizzo fisico è di *24 bit*. Qual è la dimensione dell'*indirizzo virtuale*?

La dimensione di ogni pagina è $1024B = 2^{10}\ B$, quindi sono richiesti *10 bit* per indirizzare ciascun byte di ogni pagina (*10 bit* OFFSET).

Pertanto i *24 bit* dell'indirizzo fisico, includono *10 bit* per l'OFFSET e *14 bit* per il NUMERO della pagina fisica.

Quindi ogni ingresso della *Page Table* è composto di *2 bit* per *valid* e *dirty*, e *14 bit* per indirizzare le pagine fisiche → *16 bit* in totale (*2 B*)

Il numero totale di ingressi della *Page Table* è $8\ MB / 2B = 2^{23} / 2 = 2^{22}$, indirizzabili con *22 bit*. La dimensione dell'indirizzo virtuale è quindi:

22 bit + *10 bit* (OFFSET) = *32 bit*

Memoria Virtuale e TLB

In un sistema di *memoria virtuale* ogni accesso alla memoria richiede due accessi: prima alla Page Table e quindi alla memoria fisica.

Per ridurre questo overhead si sfrutta un meccanismo detto *TLB (Translation Lookaside Buffer),* una sorta di cache della Page Table.

La TLB contiene copia delle entry della Page Table riferite recentemente, e, come la cache, può sfruttare l'associatività per ridurre i conflitti.

L'indirizzo virtuale viene scomposto in virtual page address e offset (nella pagina).
Il virtual page address viene utilizzato per accedere alla TLB come accadeva per la cache ovvero …

… se la TLB contiene 2^K set, i K bit meno significativi del virtual page address sono utilizzati per indirizzare un insieme della TLB, i bit rimanenti sono utilizzati come *tag* nelle entry della TLB.

Memoria Virtuale e TLB: Esercizio 3

Si consideri un sistema di memoria virtuale paginata, con *pagine* di *1 KB* e *indirizzo virtuale* di *32 bit*. Si consideri una *TLB 2-way-associative* di *512 B*, in cui ciascun elemento consta di *4 B* suddivisi in: *2 bit* per *valid* e *dirty*, *TAG* e corrispondente numero di pagina fisica. Calcolare la dimensione dell'*indirizzo fisico*.

Offset (o displacement) di pagina: la pagina ha dimensione *1 KB* = 2^{10} *B* e quindi servono *10 bit* per l'offset.

I rimanenti *22 bit* dell'indirizzo virtuale servono per indirizzare gli elementi della *Page Table (NUMERO di pagina virtuale)*.

22 bit per indirizzare la Page Table	**10 bit OFFSET**

Memoria Virtuale e TLB: Esercizio 3 (cont.)

Si consideri un sistema di memoria virtuale paginata, con *pagine* di *1 KB* e *indirizzo virtuale* di *32 bit*. Si consideri una *TLB 2-way-associative* di *512 B*, in cui ciascun elemento consta di *4 B* suddivisi in: *2 bit* per *valid* e *dirty*, *TAG* e corrispondente numero di pagina fisica. Calcolare la dimensione dell'*indirizzo fisico*.

22 bit per indirizzare la Page Table	10 bit OFFSET

Vediamo come l'indirizzo virtuale viene scomposto nell'accesso alla *TLB*:

tot. ingressi TLB = 512 B / 4 B = 128

essendo la *TLB 2-way associative*

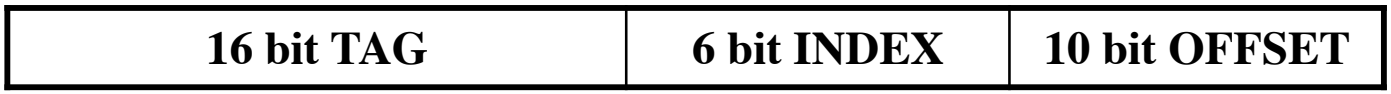
tot. insiemi TLB = 128 / 2 = 64 = 2^6

Quindi:

16 bit TAG	6 bit INDEX	10 bit OFFSET

Memoria Virtuale e TLB: Esercizio 3 (cont.)

Si consideri un sistema di memoria virtuale, con *pagine* di *1 KB* e *indirizzo virtuale* di *32 bit*. Si consideri una *TLB 2-way-associative* di *512 B*, in cui ciascun elemento consta di *4 B* suddivisi in: *2 bit* per *valid* e *dirty*, *TAG* e corrispondente numero di pagina fisica. Calcolare la dimensione dell'*indirizzo fisico*.



Per calcolare la dimensione dell'indirizzo fisico, ricordiamo che ogni entry della *TLB* è del tipo:

Dato che ogni entry comprende *32 bit*, il numero della pagina fisica occupa *32 - #(valid, dirty) - # TAG = 32 – 2 – 16 = 14*. Quindi

 # Ind. Fisico = #(num. Pagina) + #OFFSET = 14 + 10 = 24

Memoria Virtuale e TLB: Esercizio 4

Si consideri un sistema di memoria virtuale, con
- indirizzi virtuali di *40 b*
- pagine di *16 KB*
- indirizzi fisici di *26 b*

Calcolare la dimensione della tabella delle pagine, assumendo che ogni entry includa *4 bit* per valid, dirty, protezione e uso.

L'indirizzo virtuale si suddivide nel modo seguente:

26 bit per indirizzare la Page Table	14 bit OFFSET

quindi il numero di entry della Page Table (numero di pagine virtuali) è 2^{26}.
Ogni entry della Page Table ha la forma seguente:

V-D-P-U (4 bit)	Indirizzo Pagina fisica (12 bit)

Quindi complessivamente la Page Table occuperà uno spazio di $2^{26} \cdot 16$ *bit,* ovvero 2^{27} *B = 128 MB.*

Memoria Virtuale e TLB: Esercizio 4 (cont.)

Si supponga che il sistema di memoria virtuale precedente (indirizzi virtuali di *40 b,* pagine di *16 KB,* indirizzi fisici di *26 b)* sia implementato con una TLB 4-way associative di *256* elementi. Si indichi dimensione e composizione delle entry della TLB e dimensione della TLB stessa.

L'indirizzo virtuale è ora utilizzato per accedere alla TLB. I *256* elementi sono suddivisi in *64* set di quattro elementi, che richiedono *6 bit* per essere indirizzati. Dunque:

20 bit TAG	6 bit INDEX	14 bit OFFSET

quindi le entry della TLB conterranno:

V-D-P-U (4 bit)	20 bit TAG	12 bit Physical Page Address

Quindi la dimensione complessiva della TLB risulta essere *256• 36 bit,* ovvero $2^8 * 4 * 9 = 2^{10} * 9 = 9$ *Kb.*

Memoria Virtuale e TLB: Esercizio 4 (cont.)

Si supponga che il sistema di memoria precedente (indirizzi virtuali di *40 b,* pagine di *16 KB,* indirizzi fisici di *26 b)* abbia una cache di tipo set-associative a due vie con blocchi di 8 B e una dimensione totale di *4 KB* (dati). Spiegare le modalità di accesso alla memoria (struttura e dimensione delle varie parti degli indirizzi).

L'indirizzo virtuale viene utilizzato per accedere alla TLB e, nel caso di TLB-miss, alla Page Table. Una volta risolti i miss verrà generato un indirizzo in memoria fisica.

L'indirizzo fisico viene quindi utilizzato per accedere alla cache. Il numero di blocchi della cache è

$$dim.\ tot.\ /\ dim.\ blocco = 4 * 2^{10} / 8 = 2^{9}$$

e quindi si avranno $2^9 / 2 = 2^8$ set costituiti da due blocchi.

Pertanto l'indirizzo fisico è strutturato nel modo seguente (#tag = IND_FIS - OFFSET – INDEX = 26 – 3 – 8 = 15):

15 bit TAG	8 bit INDEX	3 bit OFFSET