# Schema di principio di una Cache

Processore tipo z64 e memoria organizzata con un banco a 32 bit (4 byte)

Si ipotizza che ci sia sempre allineamento dei byte e che si accedano sempre a 4 byte

Di seguito si ipotizzerà che le memorie cache vengono realizzate, come avviene normalmente, con memorie RAM STATICHE, mentre la memoria di lavoro è realizzata con RAM DINAMICHE.

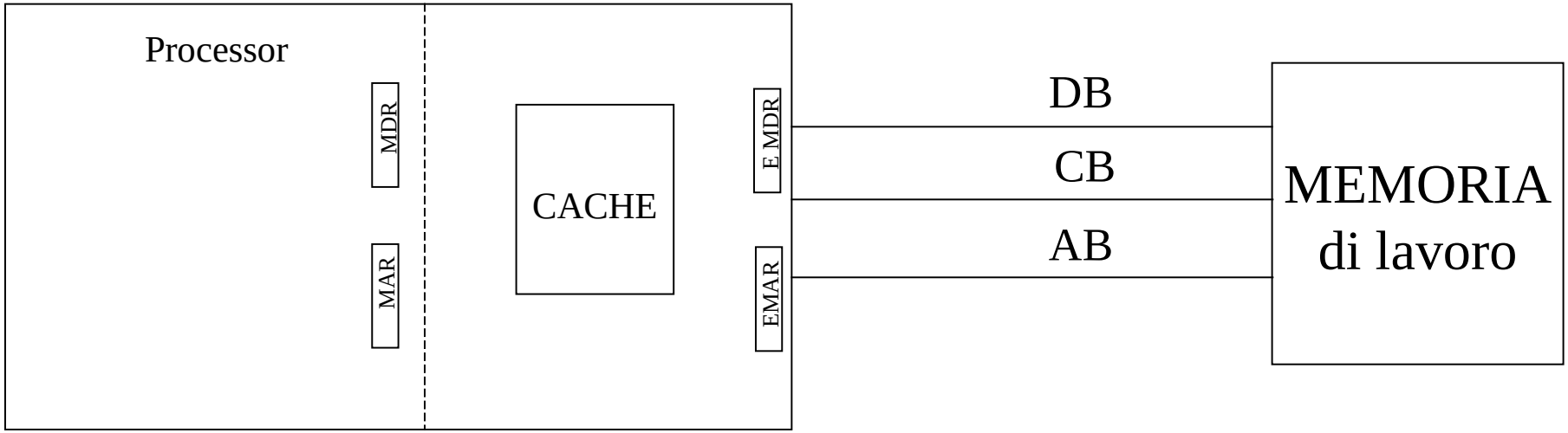
Per capire il funzionamento delle cache non è necessario conoscere nel dettaglio il funzionamento delle memorie dinamiche, si deve solo ricordare che le modalità di lettura e scrittura sono identiche a quelle delle memorie statiche, ma che sono solo più lente.

Inoltre per semplicità di presentazione e per non appensantire il layout dei disegni, si ipotizzerà che:

- il processore possa generare al più 32 bit di indirizzi, questo non ostacola l'apprendimento della metodologia, in quanto l'estensione a 48 o a 64 bit è immediata;
- all'interno della memoria ci sia sempre l'allineamento delle parole;
- lo spazio di indirizzamento della memoria di lavoro è pari alla massima che il processore possa generare. Dopodiché criticheremo questa ipotesi ed introdurremo una modalità che consente di estendere lo spazio di indirizzamento, generabile dai processori, utilizzando memorie di massa, quali i solid state device (SSD, dispositivi conosciuti anche come memorie flash) o i dischi.

Di seguito si farà vedere prima la soluzione più semplice, in cui si ipotizza di utilizzare solo un livello di cache e poi successivamente faremo vedere soluzioni in cui ci sono più livelli di cache, senza e con la presenza di dispositivi dedicati all'interazione con la memoria di massa

Nel caso di presenza di solo memoria cache di primo livello, si ipotizzerà che venga allocata nello stesso chip del processore. Per salvaguardare le funzionalità del SCA del processore, come visto fino ad ora, si cambiano solo le funzionalità dei registri di interfaccia MAR e il MDR. Mentre prima servivano per interconnettere il SCA dello z64 con la memoria di lavoro, ora servono ad interconnettere il SCA del processore con la memoria cache, mentre la memoria cache sarà interconnessa alla memoria di lavoro con altri due registri di interfaccia: EMAR (External Memory Address Register) ed EMDR (External Memory Data Register), che servono per interconnettere il processore con i bus della memoria di lavoro. Come nel caso senza cache il MAR e il MDR servono, rispettivamente, a memorizzare l'indirizzo di memoria da cui leggere o su cui scrivere il dato e a memorizzare il dato in uscita o in ingresso verso la memoria. Naturalmente si accederà in memoria di lavoro solo in caso di miss, mentre in caso di hit il trasferimento riguarderà unicamente la memoria cache. In Figura è schematizzata l'architettura di massima a cui faremo riferimento per progettare la memoria cache di primo livello in cui sono evidenziati i registri di interfaccia.



Il SCO, nel caso di presenza della cache di primo livello nel chip del processore, sarà diverso da quello visto precedentemente, in quanto in caso di ***hit*** dovrà trasferire i dati da/nella cache invece che da/nella memoria di lavoro, mentre nel caso di ***miss*** dovrà accedere alla memoria di lavoro per aggiornare la cache, inoltre in quest'ultimo caso, per garantire la coerenza tra la cache e la memoria di lavoro, si prevede l'utilizzazione della politica di ***write-back*** e, quindi, prima di sovrascrivere nella memoria cache i nuovi dati sarà necessario prima memorizzare il contenuto del blocco interessato alla lettura/scrittura nella memoria di lavoro.

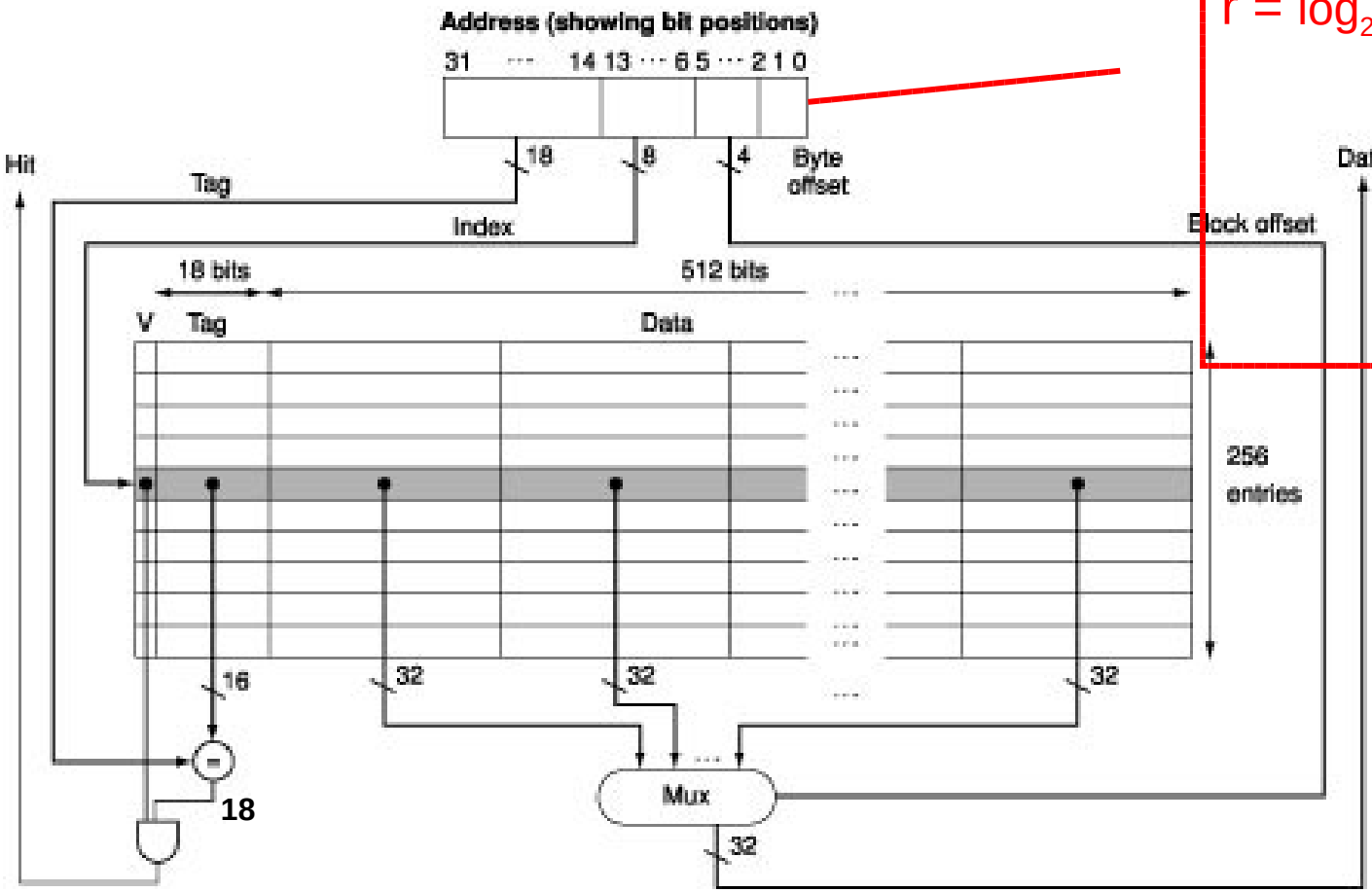
Data la differente modalità delle interazioni, nel prosieguo di questo Capitolo, si presenteranno quelle parti del SCA e del SCO del processore interessati prima alla lettura/scrittura dei dati nella cache nel caso di hit e poi alla lettura/scrittura dei dati nel caso di miss. In particolare, come detto, data la politica di «write-back» si prevede quindi prima una scrittura nella memoria di lavoro di tutto il blocco occupante la cache dove dovrà essere inserito il blocco interessato alla lettura/scrittura. Dopodiché ci sarà una sovrascrittura del blocco interessato alla lettura/scrittura nella cache sul blocco già trasferito in memoria di lavoro. A questo punto si sono create le stesse condizioni logiche di accesso alla cache nel caso di hit e quindi lo SCO può completare il trasferimento del dato da uno dei registri interni del processore verso la memoria cache o viceversa.

Poiché il SCO deve generare segnali di lettura e scrittura verso la memoria e vero la cache, per semplicità di lettura del lettore si indicheranno con $\boldsymbol{MR}$ e $\boldsymbol{MW}$ i segnali di controllo verso la memora e con $\boldsymbol{MR_C}$ e $\boldsymbol{MW_C}$ i segnali di controllo per la lettura e scrittura dei moduli di memoria della cache.

# Esempio di cache con blocchi da 16 parole (con parola da 4 byte)

- Nel presentare come è possibile realizzare una memoria cache di primo livello si ipotizzerà di utilizzare una cache ad ***accesso diretto*** con blocchi di 16 parole, in cui ogni parola è di 4 byte, come quella del processore Intrisity FastMATH, un processore embedded basato sull'architettura MIPS. Quindi in ogni riga della cache, oltre al flag di validità e il tag, verranno memorizzate 16 parole da 4 byte ciascuna.
- In Figura è schematizzata l'organizzazione logica di questa cache. La cache schematizzata è di 16 KB.
- Dato che la cache che si andrà a progettare è di 16 KB, indirizzabile con 32 bit (n=32), con 256 blocchi, di cui ogni blocco è 16 parole e ogni parola è di 4 B, pertanto si avrà:
  - s = 8, dato che $s = \log_2(16KB/(16 \times 4B))$,
  - r = 6, dato che $r = \log_2(64)$, di cui:
    - 4 bit identificano la parola all'interno del blocco, e
    - 2 il singolo byte all'interno della singola parola
  - tag = 18 bit, dato che tag = n-s-r
- Dalla Figura si può osservare che per accedere in lettura alla memoria cache si utilizza il campo index dell'indirizzo per accedere al blocco, si verifica che il campo flag sia asserito e che il tag dell'indirizzo (di 18 bit) coincide con quello memorizzato nella riga selezionata. In caso di hit il block offset (i 4 bit di selezione del mux) identificano il dato da leggere.

# Esempio di cache con blocchi da 16 parole (con parola da 4 byte)




n = 32

$s = \log_2(16KB/64B) = 8$

$r = \log_2(64) = 6$,

di cui

4 identificano la parola all'interno del blocco,

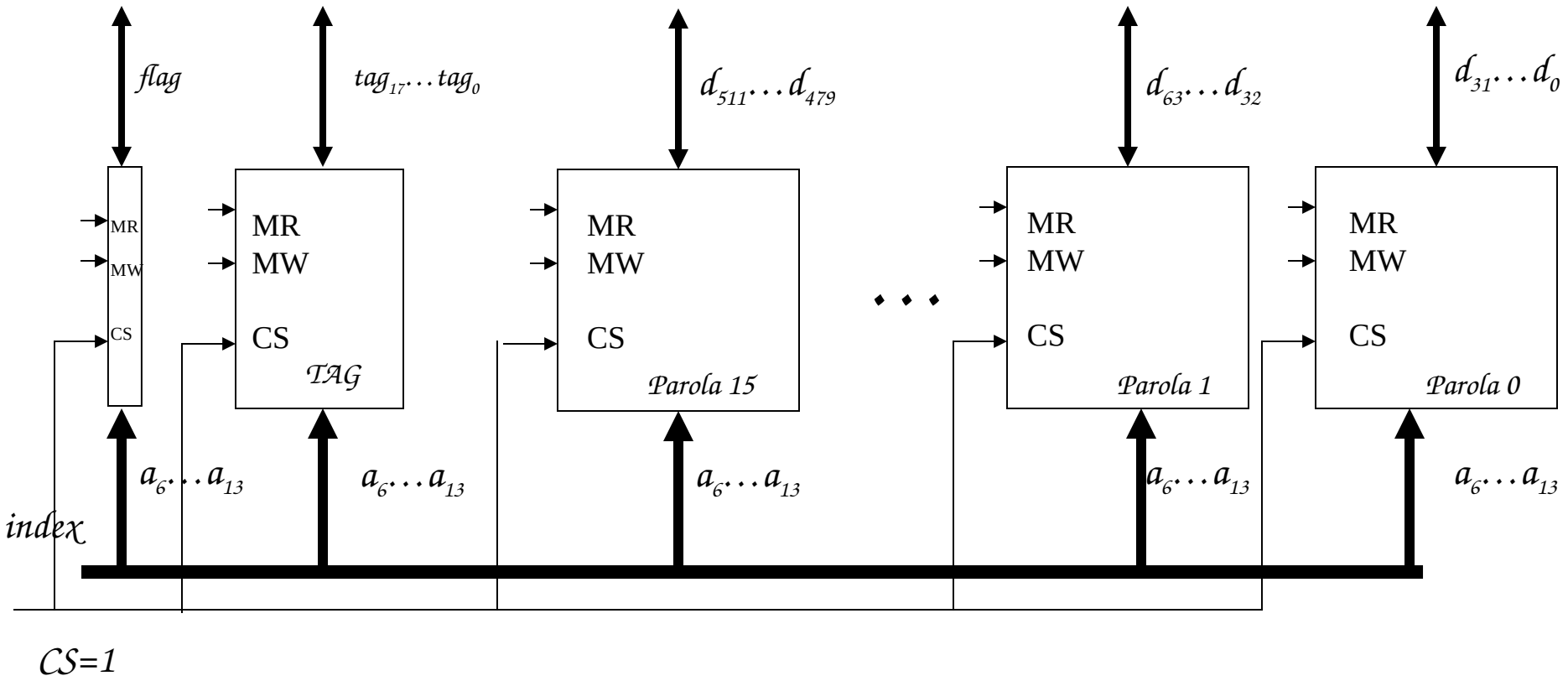
2 il byte all'interno della singola parola

I bit del modulo di flag vengono messi tutti a ZERO in fase di accensione del sistema e vengono messi ad UNO mano a mano che vengono caricate le long word prese dalla memoria di lavoro. Da notare che in caso di presenza del ***S.O.*** tali flag verrebbero posti a zero ogni qual volta ci fosse l'attivazione di un altro processo.

POSSIAMO DIRE CHE LA MODALITA' DI FUNZIONAMENTO CHE PREVEDIAMO è QUELLA DEL KERNEL, IN QUANTO TUTTI I DATI DEL KERNEL SONO NELLA MEMORIA DI LAVORO E QUINDI è COME SE LO SPAZIO DI INDIRIZZAMENTO DELLA MEMORIA FOSSE QUELLO DI DIMENSIONE MASSIMA ($2^{32}$ in questo caso)

# Possibile organizzazione di una cache ad accesso diretto con blocchi da 16 parole (ogni parola da 4 byte)

Per la progettazione si ipotizzerà di avere a disposizione memorie statiche indirizzabili a 8 bit e, quindi, con $2^8$ «righe», come schematizzato in Figura. Il numero di colonne di ogni singolo modulo di memoria invece dipende da ciò che memorizza. Pertanto si prevede di utilizzare un modulo di memoria da un bit per memorizzare i flag, di 18 bit per memorizzare i tag e di 32 bit per memorizzare le parole dei blocchi. In Figura è schematizzata una possibile architettura. Da notare che l'index, che identifica il possibile blocco dove potrebbe essere memorizzato il dato da leggere/scrivere, è utilizzato per identificare la riga di tutte le memorie statiche utilizzare. Gli ingressi/uscite dei dati di ogni singolo modulo di memoria, a differenza degli indirizzi, invece, sono connessi in modo distinto, pertanto in Figura ? le linee dati dei vari moduli di memoria sono identificati in modo differente. Nelle prossime figure verrà descritto come tali connessioni sono utilizzate per trasferire dati con la memoria di lavoro e con il registro di interfaccia del processore, che serve per disaccoppiare la parte dei calcolo del SCA del processore con la cache.



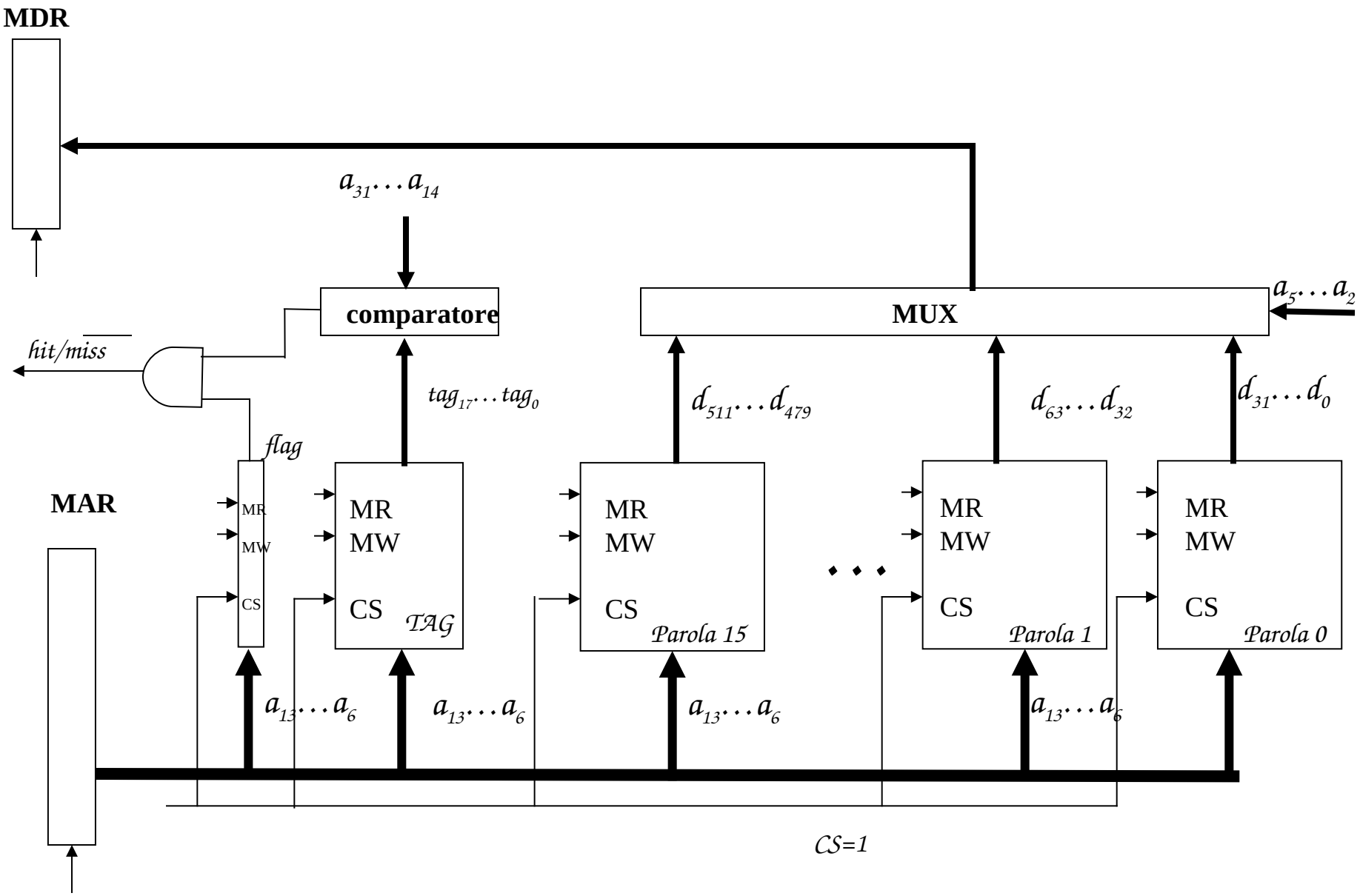
## *SCHEMA SCA per LETTURA dalla CACHE*

In Figura ? è schematizzata l'architettura della cache di I livello per supportare la lettura di un dato dalla cache verso lo SCA del processore, quindi le connessioni disegnate servono solo per permettere il trasferimento dati dalla cache verso il processore, da ricordare che il SCO per la lettura di un dato scriverà l'indirizzo del dato nel registro di interfaccia MAR e che il dato letto, prima di essere scritto in uno dei registri interni verrà memorizzato nel registro MDR..

Quindi al normale microprogramma del processore per accedere alla memoria si introduce uno stato in cui una volta trasferito l'indirizzo a cui si vuole accedere sul MAR e generato il CS e MR per i moduli di memoria si verifica se c'è hit o meno:

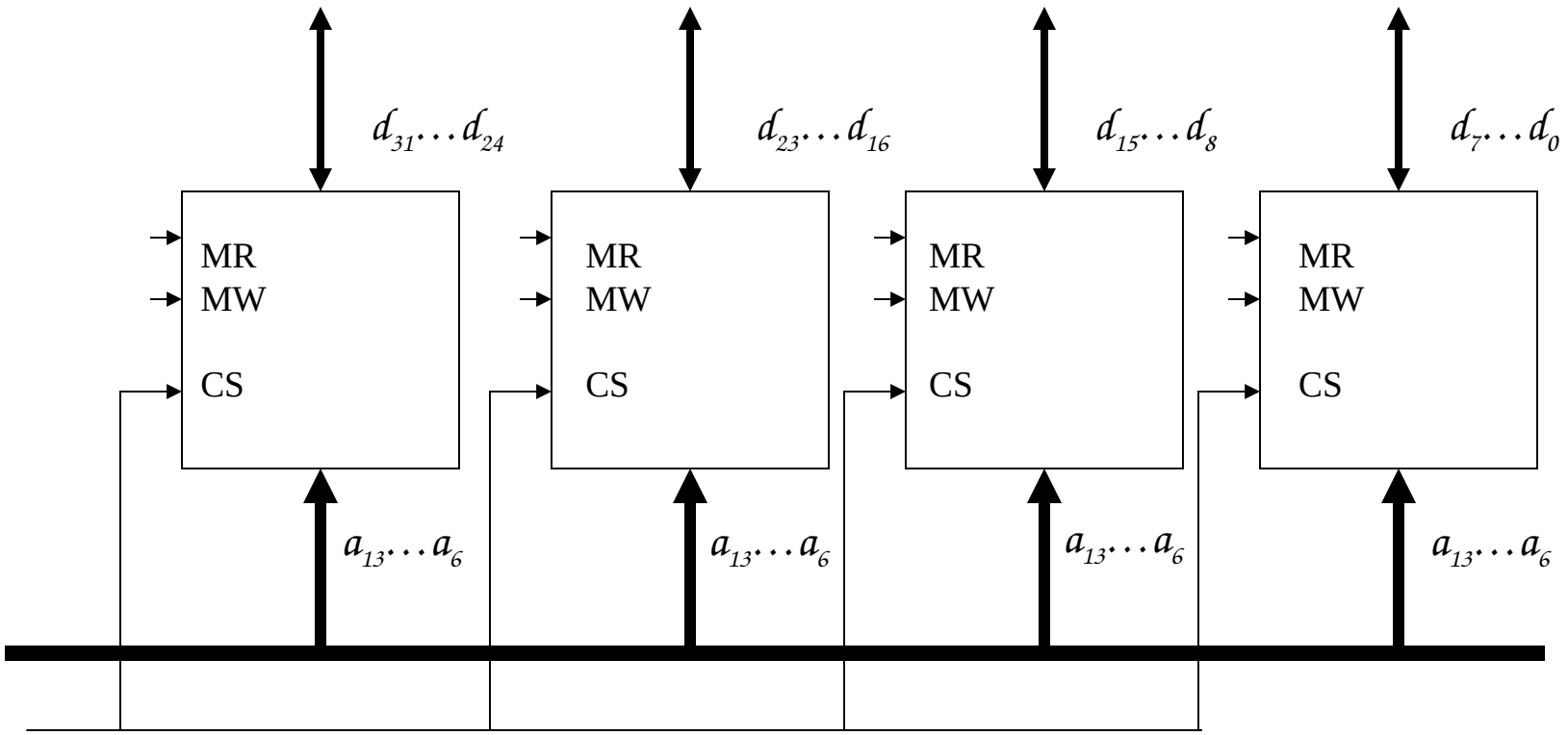
•- nel caso di **hit** (flag di riga uguale ad 1 e valore del tag del blocco identico a quello memorizzato nei bit $a_{31}$ …$a_{24}$ del MAR) si conclude il normale ciclo di lettura, cioè il SCO asserirà il segnale di controllo di abilitazione alla scrittura sul registro MDR, Da notare che nel frattempo i bit $a_5$ …$a_2$ del MAR avranno selezionato il dato relativo all'indirizzo di memoria a cui si voleva accedere; il SCO a questo punto potrà completare il trasferimento del dato in uno dei registri interni del processore.

-nel caso di **miss** (hit/miss negato = 0) il segnale di controllo MR generato dallo SCO viene posto dallo SCO stesso a ZERO, che lo rimetterà ad UNO solo dopo aver recuperato il blocco dati dalla memoria. In tal caso il SCO dovrà accedere alla memoria di lavoro per prelevare il blocco di dati da sostituire a quelli nella cache, però prima di sostituire, scrive i dati del blocco dalla cache nella memoria di lavoro. Una volta trasferito il blocco dei dati dalla memoria alla cache si potrà concludere la lettura ( il segnale da miss diverrà hit). Il trasferimento dati dalla cache verso memoria e viceversa verrà analizzato successivamente nei paragrafi ? e ?.


MDR
$a_{31}\ldots a_{14}$
comparatore
MUX
$a_5\ldots a_2$
hit/miss
$tag_{17}\ldots tag_0$
$d_{511}\ldots d_{479}$
$d_{63}\ldots d_{32}$
$d_{31}\ldots d_0$
flag
MAR
MR
MW
CS
TAG
Parola 15
Parola 1
Parola 0
$a_{13}\ldots a_6$
CS=1

I moduli di memoria in cui sono memorizzati i dati della cache sono memorie statiche di 32 bit, quindi possono essere realizzati utilizzando 4 banchi di memoria ciascun di 8 bit, come schematizzato in Figura, dove è rappresentata l’organizzazione interna del primo modulo di memoria.

# Memoria di 32 bit: organizzazione con moduli di un byte



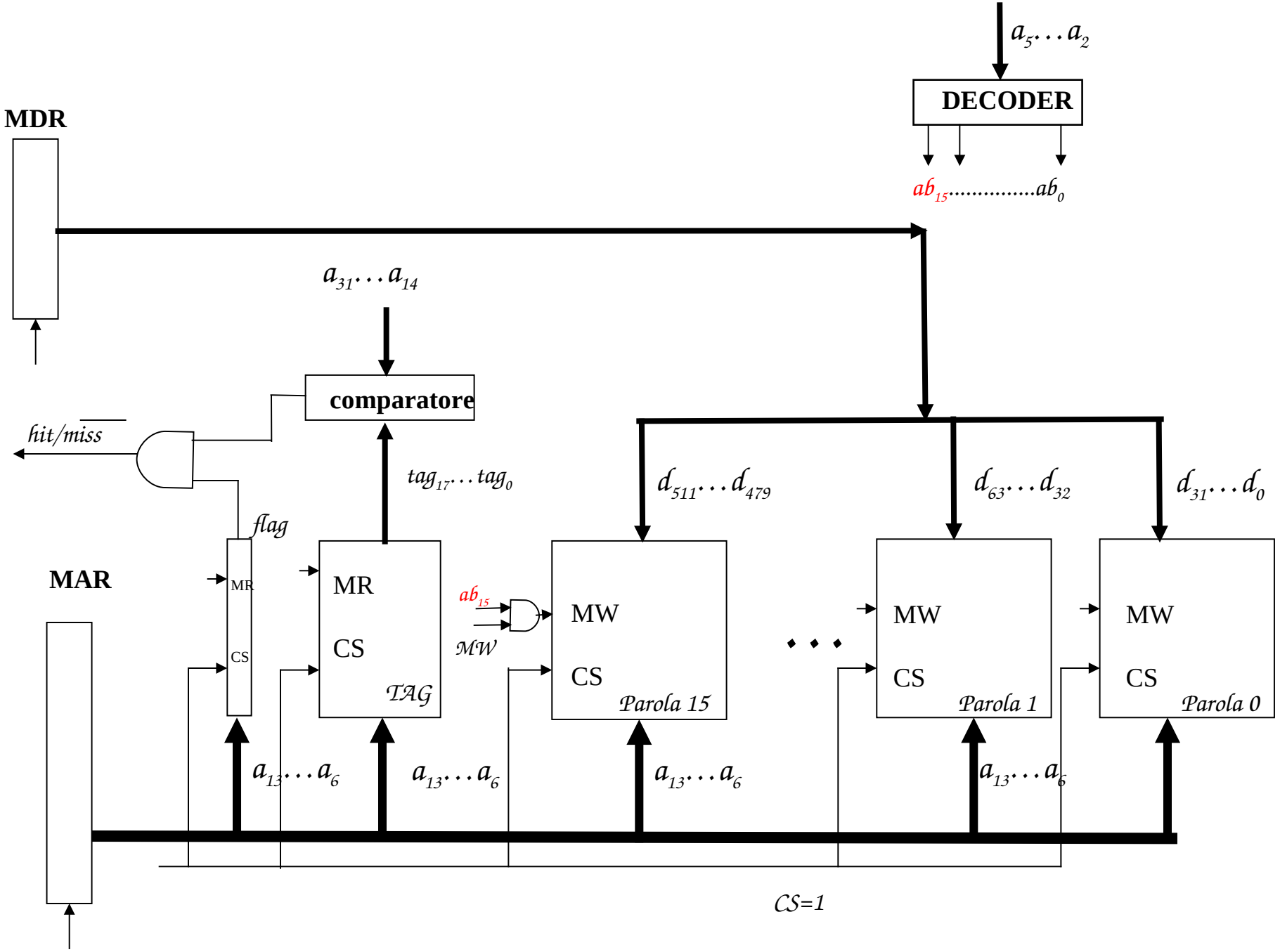
## *SCHEMA SCA per SCRITTURA sulla CACHE dal processore*

In Figura ? è schematizzata l'architettura della cache di I livello per supportare la scrittura di un dato dallo SCA del processore alla cache, quindi le connessioni disegnate servono solo per permettere il trasferimento dati dal processore verso la cache. Nel disegno è rappresentato una sola porta AND che pilota i moduli di memoria da 32 bit. Il generico modulo i-esimo sarà abilitato alla scrittura solo quando oltre al segnale WR, proveniente dal SCO, sarà presente nei bit bit $a_5$ …$a_2$ del MAR la codifica dell'i-esimo modulo. A tal fine usiamo un decoder che a fronte del valore dei bit $a_5$ …$a_2$ genererà in uscita un 1 solo nell'uscita codificata da tali bit, mentre le rimanenti 15 uscite saranno pari a 0.

E' da ricordare che il SCO per la scrittura di un dato scriverà l'indirizzo del dato nel registro di interfaccia MAR e che il dato da scrivere, prima di essere scritto nella cache verrà memorizzato nel registro MDR..
Quindi al normale microprogramma del processore per accedere alla memoria si introduce uno stato in cui una volta trasferito il dato da scrivere nell'MDR e l'indirizzo a cui si vuole accedere sul MAR e generato il CS e MR per verificare se c'è un hit o meno, notare come il MR è connesso solo per il modulo dei flag e dei tag.

-nel caso di **hit** (flag di riga uguale ad 1 e valore del tag del blocco identico a quello memorizzato nei bit $a_{31}$ …$a_{24}$ del MAR) si conclude il normale ciclo di scrittura, cioè il SCO asserirà il segnale di controllo di abilitazione alla scrittura sul MW. Da notare che nel frattempo i bit $a_5$ …$a_2$ del MAR tramite il decoder e una porta AND abiliteranno in scrittura solo il modulo di memoria dove è da memorizzare il dato relativo all'indirizzo di memoria a cui si voleva accedere;

-nel caso di **miss** (hit/miss negato = 0) il SCO dovrà accedere alla memoria di lavoro per prelevare il blocco di dati da sostituire a quelli nella cache, però prima di sostituire, dovrà scrivere i dati del blocco dalla cache nella memoria di lavoro. Una volta trasferito il blocco dei dati dalla memoria alla cache si potrà concludere la scrittura (il segnale da miss diverrà hit). Il trasferimento dati dalla cache verso memoria e viceversa verrà analizzato successivamente nei paragrafi ?? e ???.


$a_5 \ldots a_2$
DECODER
MDR
$ab_{15} \ldots\ldots\ldots\ldots ab_0$
$a_{31} \ldots a_{14}$
comparatore
hit/miss
$tag_{17} \ldots tag_0$
$d_{511} \ldots d_{479}$
$d_{63} \ldots d_{32}$
$d_{31} \ldots d_0$
flag
MAR
MR
CS
MR
CS
TAG
$ab_{15}$
MW
MW
CS
Parola 15
MW
CS
Parola 1
MW
CS
Parola 0
$a_{13} \ldots a_6$
$a_{13} \ldots a_6$
$a_{13} \ldots a_6$
$a_{13} \ldots a_6$
CS=1

***SCHEMA SEMPLIFICATO per la SCRITTURA DATI in CACHE – LETTURA DALLA MEMORIA***

Nella Figura ??? È rappresentata quella parte del SCA della memoria cache di I livello necessaria per poter leggere i dati dalla memoria di lavoro e memorizzare i dati di un blocco nella cache. Si ricorda che la lettura dei dati dalla memoria avviene nel caso di miss in lettura o in scrittura.

Il SCO del processore per trasferire un blocco dei dati dalla cache verso la memoria esegue tante letture in memoria per quante sono le parole (dati) del blocco da aggiornare (16 in questo caso). A tal fine utilizza un contatore modulo 16, che permette di tenere in conto il numero dei dati del blocco da trasferire. AD ogni trasferimento il SCO del processore incrementa il contatore, quando torna a 0, evidenziato dal segnale TC, è terminata la fase di trasferimento. Notare che gli indirizzi di memoria da cui leggere i dati sono a loro volta consecutivi e per ogni blocco da aggiornare è necessario leggere i dati memorizzati a partire dalla locazione di memoria:

- $a_{31} \ldots a_6$ 0 0 0 0

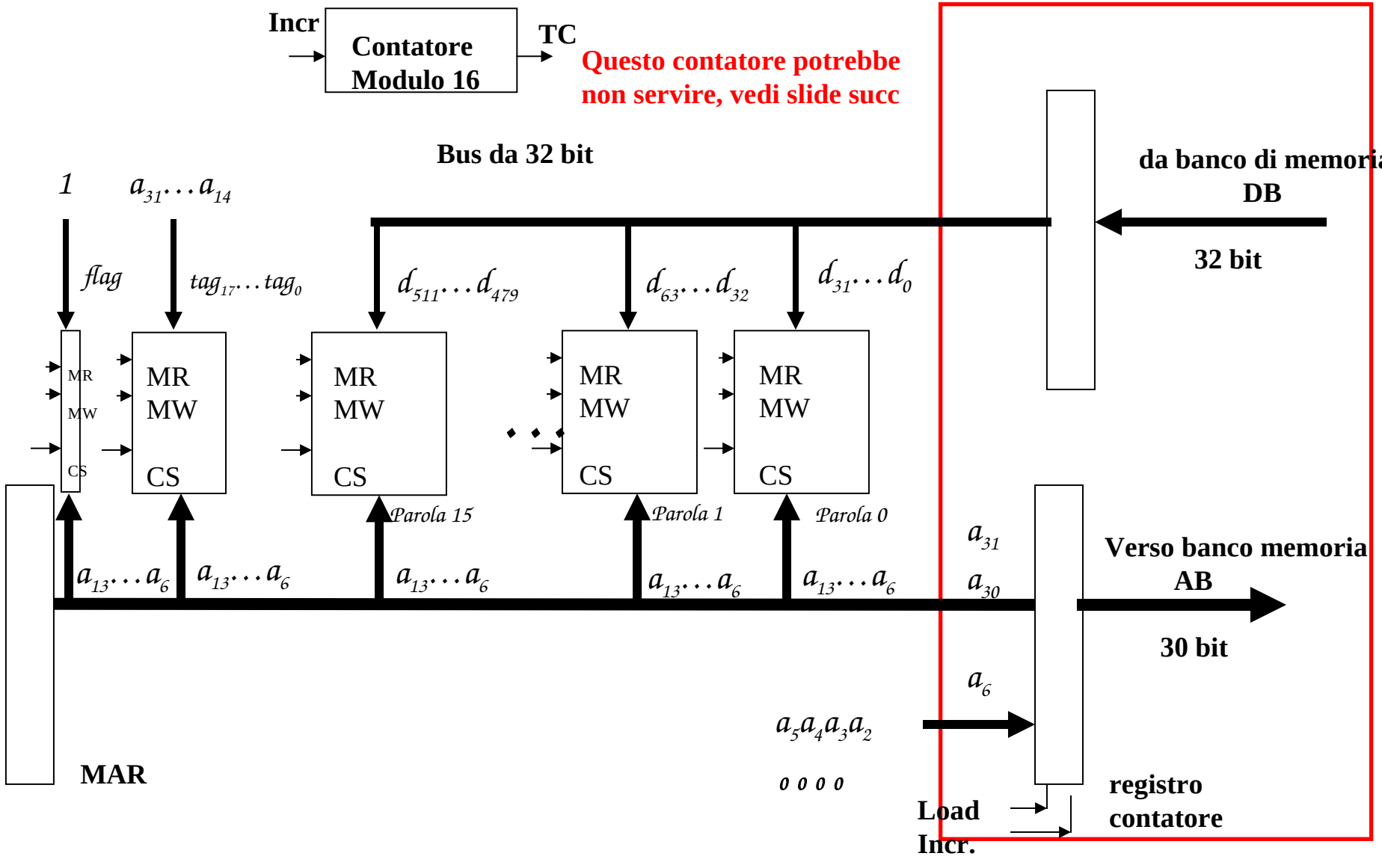
fino a:

- $a_{31} \ldots a_6$ 1 1 1 1

dove il valore dei bit $a_5 \ldots a_2$ devono essere generati a loro volta da un contatore.

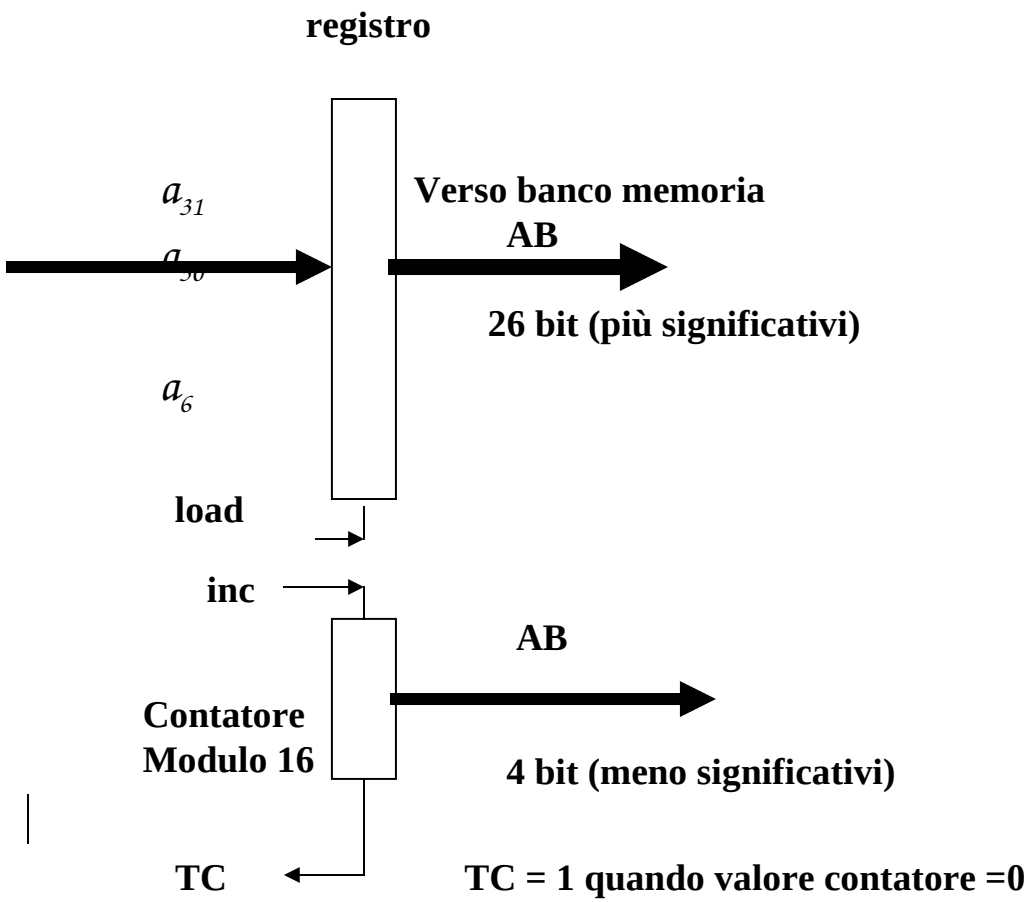
Quando il TC varrà 1 il SCO del processore potrà scrivere nel modulo TAG l'identificativo caratterizzante il blocco ora memorizzato nella memoria cache. Per poter effettuare ciò è necessario che i bit $a_{31} \ldots a_{14}$, memorizzati nel MAR vengano memorizzati nella riga del modulo TAG in corrispondenza della riga codificata dai bit $a_{13} \ldots a_6$ del MAR.

Dopodiché il SCO andrà a verificare il flag di hit sarà pari ad 1 e così potrà completare il ciclo di lettura verso la cache.


Incr
Contatore Modulo 16
TC
Questo contatore potrebbe non servire, vedi slide succ
Bus da 32 bit
da banco di memoria
DB
32 bit
1
$a_{31}\ldots a_{14}$
flag
$tag_{17}\ldots tag_0$
$d_{511}\ldots d_{479}$
$d_{63}\ldots d_{32}$
$d_{31}\ldots d_0$
MR
MW
CS
Parola 15
Parola 1
Parola 0
$a_{13}\ldots a_6$
$a_{31}$
$a_{30}$
Verso banco memoria
AB
30 bit
$a_6$
$a_5a_4a_3a_2$
0 0 0 0
MAR
registro contatore
Load
Incr.

# Ottimizzazione del registro contatore con contatore modulo 16

dato che i due contatori della figura precedente devono essere aggiornati dallo SCO ad ogni lettura è opportuno utilizzare un unico contatore, come schematizzato in Figura ????.



# SCO del ciclo scrittura in cache da memoria di lavoro

**Ricapitolando, le attività del SCO del processore per poter trasferire dati dalla memria di lavoro alla cache nel caso di miss sono:**

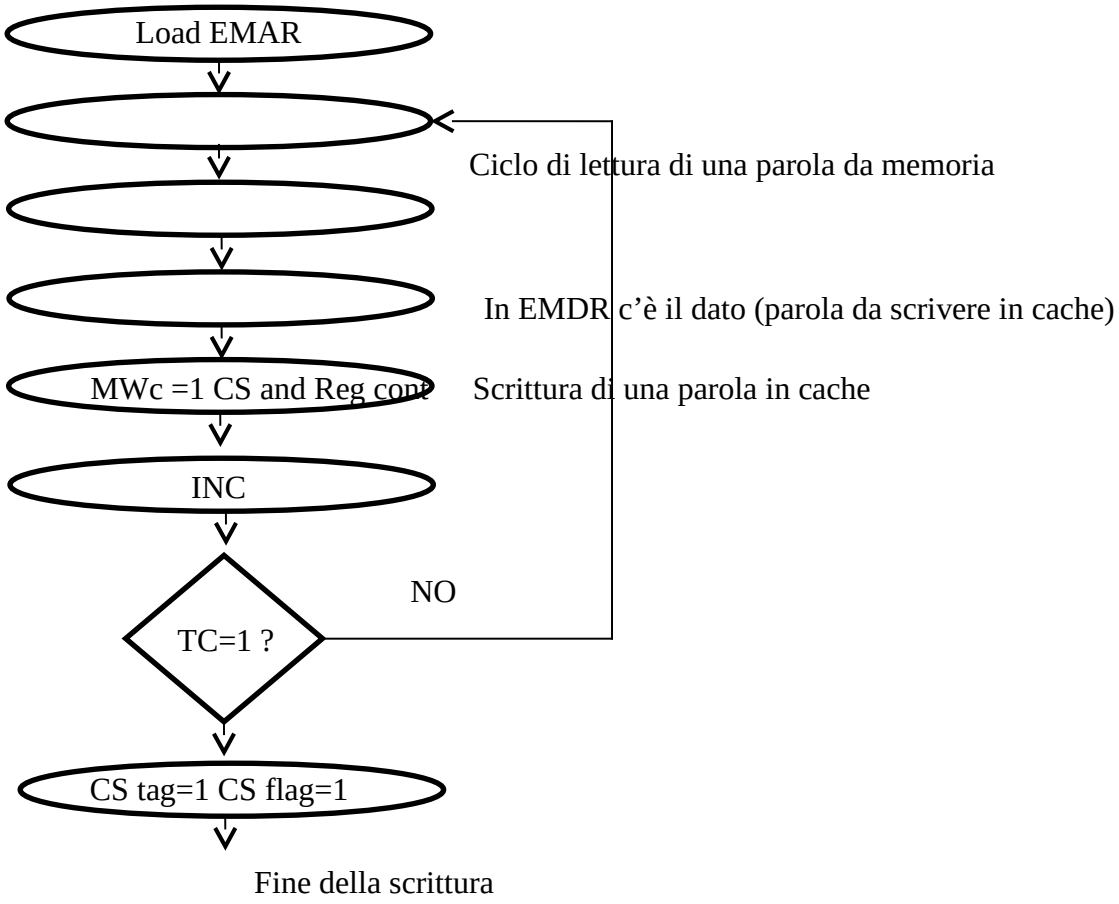
**-**

**-**

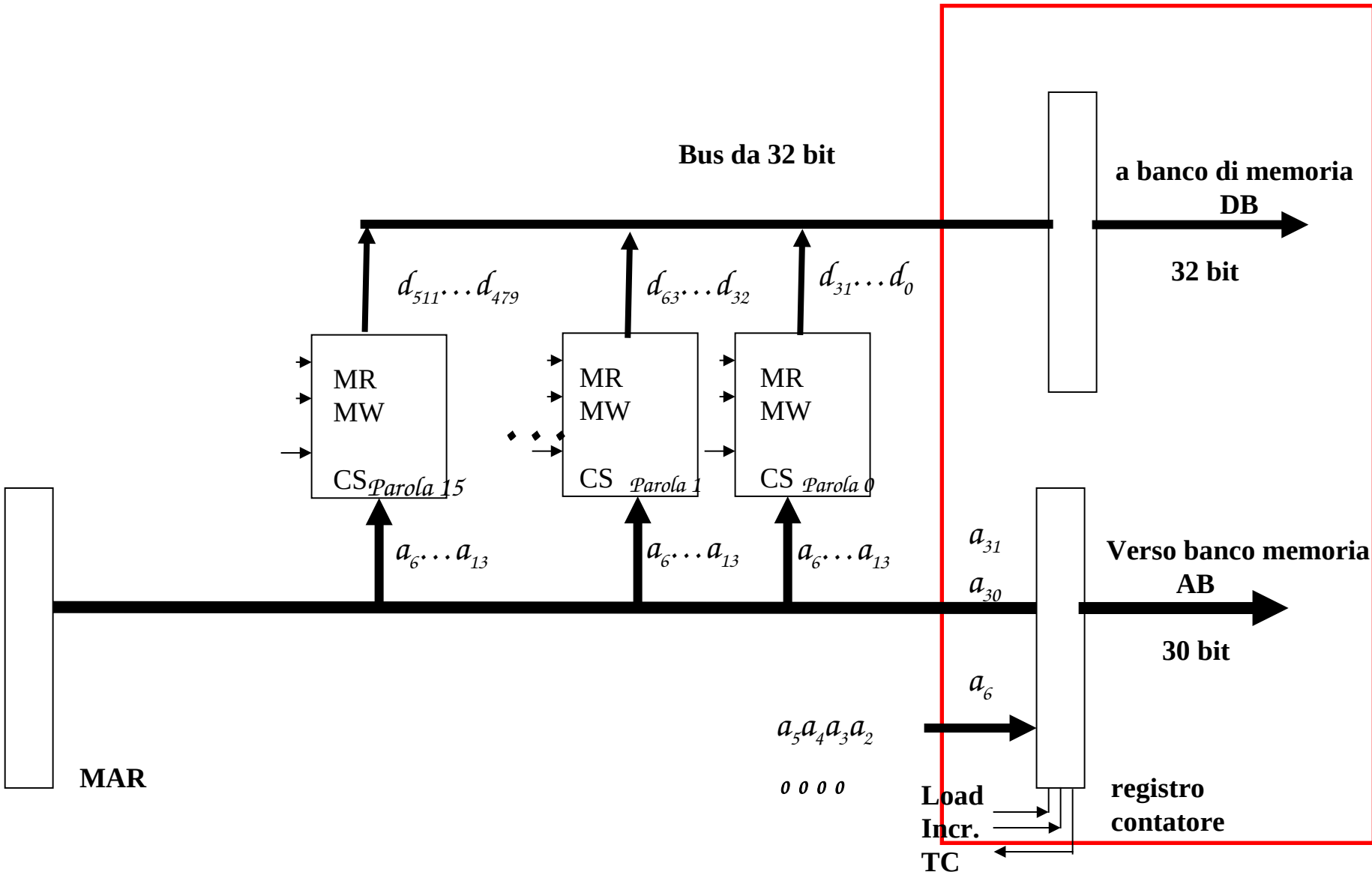
**-**

**Tali attività possono essere rappresentate dalla macchina a stati finiti schematizzata in Figura ???’**

# Ciclo scrittura in cache da memoria di lavoro (da ricontrollare)



Naturalmente, come nel caso di lettura dei dati dalla memoria e scrittura sulla cache, il valore dei bit $a_5 \ldots a_2$ possono essere generati da un contatore. In Figura ??? È rappresentata quella parte di SCA della cache interessata alla scrittura dei dati dalla cache alla memoria di lavoro. Si ricorda che questo trasferimento si rende necessario nel caso di miss in lettura o in scrittura, IN QUANTO BISOGNA MEMORIZZARE I DATI DALLA CACHE VERSO LA MEMORIA PRIMA DI SOVRASCRIVERE.



***SCHEMA SEMPLIFICATO SCRITTURA DATI DALLA CACHE NELLA MEMORIA***

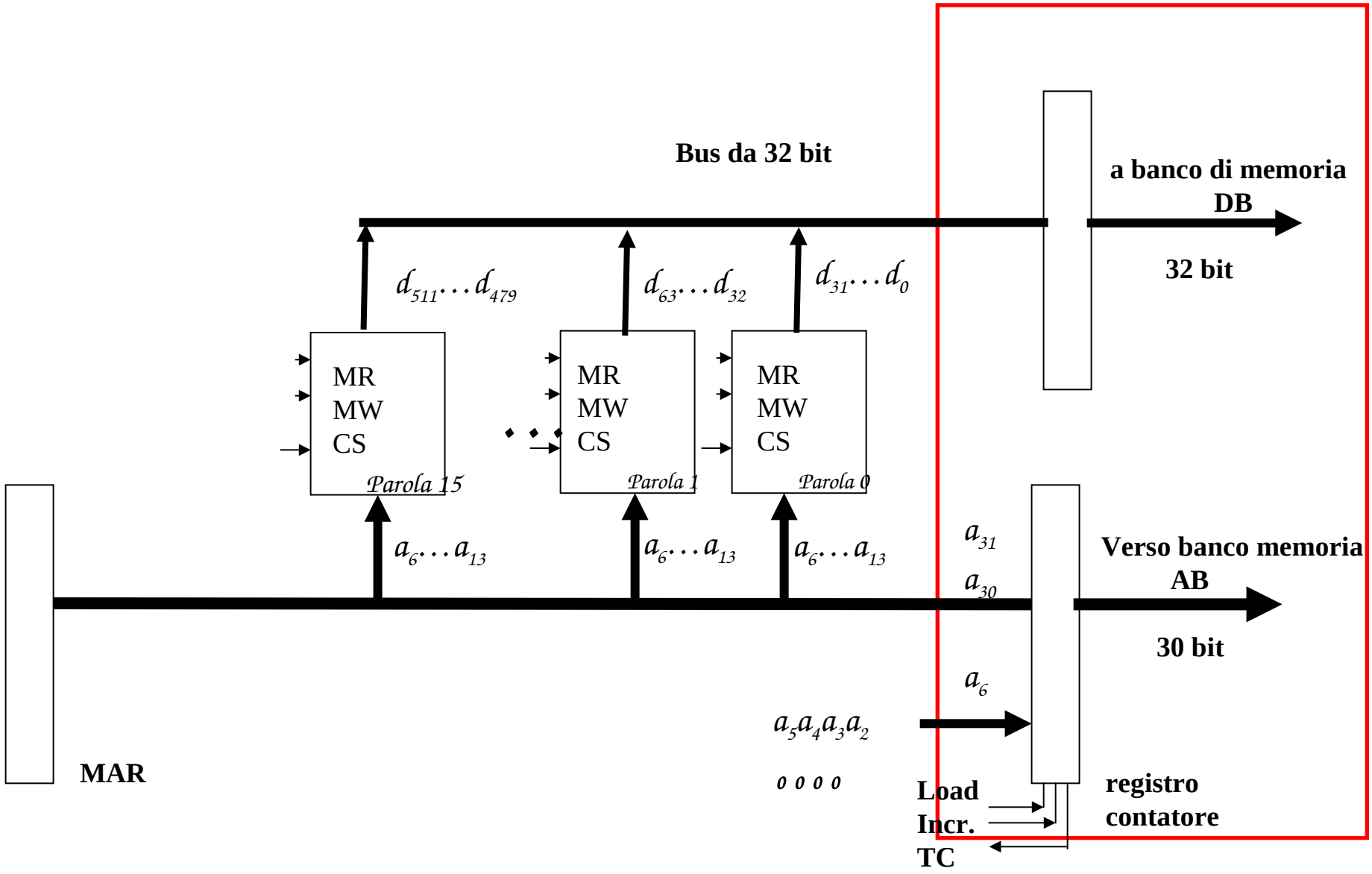
Per poter trasferire un blocco di dati dalla cache alla memoria è necessario scrivere in memoria di lavoro i 16 dati del blocco a partire dalla locazione
di memoria

- $a_{31} \ldots a_6$ 0 0 0 0

fino a:

- $a_{31} \ldots a_6$ 1 1 1 1

Naturalmente, come nel caso di lettura dei dati dalla memoria e scrittura sulla cache, il valore dei bit $a_5 \ldots a_2$ possono essere generati
da un contatore. In Figura ??? È rappresentata quella parte di SCA della cache interessata alla scrittura dei dati dalla cache alla memoria di lavoro.
Si ricorda che questo trasferimento si rende necessario nel caso di miss in lettura o in scrittura, IN QUANTO BISOGNA MEMORIZZARE
I DATI DALLA CACHE VERSO LA MEMORIA PRIMA DI SOVRASCRIVERE.


Bus da 32 bit
a banco di memoria
DB
32 bit
$d_{511}\ldots d_{479}$
$d_{63}\ldots d_{32}$
$d_{31}\ldots d_{0}$
MR
MW
CS
Parola 15
Parola 1
Parola 0
$a_6\ldots a_{13}$
$a_{31}$
$a_{30}$
Verso banco memoria
AB
30 bit
$a_6$
$a_5a_4a_3a_2$
0 0 0 0
MAR
Load
Incr.
TC
registro
contatore

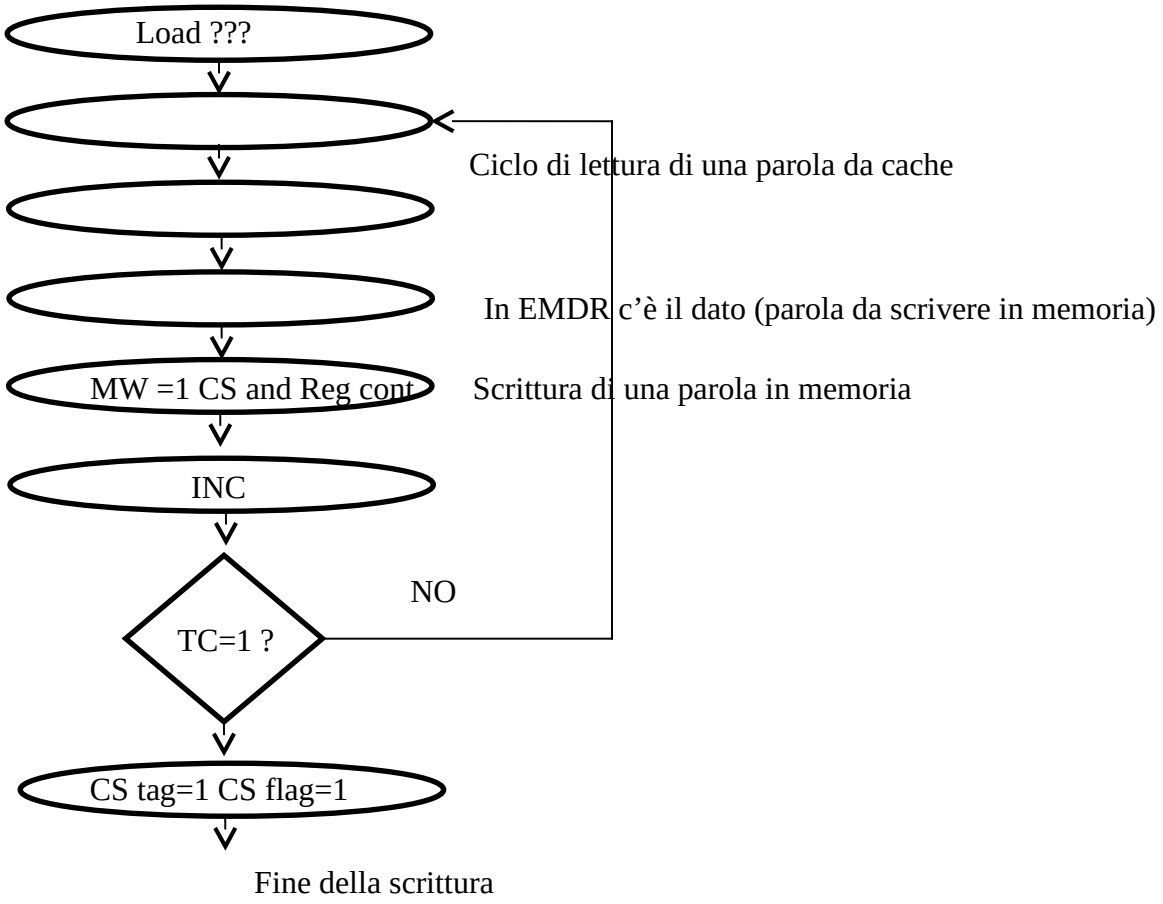
# SCO del ciclo scrittura dalla cache da memoria di lavoro

**Ricapitolando, le attività del SCO del processore per poter trasferire dati dalla cache alla memoria di lavoro nel caso di miss sono:**

-
-
-

**Tali attività possono essere rappresentate dalla macchina a stati finiti schematizzata in Figura ???’**

# Ciclo scrittura dalla cache alla memoria di lavoro (da ricontrollare)



# CONSIDERAZIONI SULLA CACHE NEL CASO DI ESECUZIONE DELLE INS E OUTS

Il trasferimento dati dalla cache alla memoria di lavoro potrebbe essere effettuato dal processore anche nel caso di esecuzione di una istruzione INSx o di una OUTSx, dato che le istruzioni prevedono il trasferimento di un blocco di dati le cui dimensioni sono memorizzate nel registro ***rcx***, mentre l'indirizzo iniziale da cui scrivere o leggere i dati è memorizzato nel registro ***dx***. Da ricordare che il formato del dato (Byte, Word, Longword, Quadword) è dato dal suffisso ***x*** specificato nell'istruzione stessa. Quindi se nella cache non sono presenti tutti o in parte gli indirizzi di memoria interessati, prima del trasferimento da o verso la periferica, è necessario trasferire i dati dalla cache verso la memoria che potrebbero essere sovrascritti.

Quindi nel caso di esecuzione dell'istruzione INSx, se l'indirizzo iniziale del blocco richiesto non è presente nella cache, ci si comporterà come nel caso di miss per la scrittura di un dato nella cache, con l'accortezza che in questo caso si dovranno trasferire in cache tanti dati per quanto è il valore memorizzato nel registro ***rcx***. Quindi, nel caso di miss in scrittura su una locazione della cache relativa ad un indirizzo di memoria non presente nella cache stessa, prima di sovrascrivere sulla riga della cache, sarà necessario trasferirne il suo contenuto nella memoria di lavoro. Naturalmente questa attività potrà essere svolta per più volte se il valore contenuto nel registro ***rcx***, dove è memorizzato il numero dei dati da trasferire dalla porta esterna, è maggiore del numero delle locazioni di cache presenti su ogni sua singola riga.

Naturalmente, anche nel caso di esecuzione dell'istruzione OUTs, eventualmente si dovranno eseguire trasferimenti dalla memoria di lavoro alla cache, se tutti o parte dei dati da trasferire verso la porta di uscita non sono presenti nella cache. Anche in questo caso per evitare sovrascritture, sarà eventualmente necessario, trasferire prima i dati dei blocchi dati della cache nella memoria di lavoro le cui locazioni dovranno essere sovrascritte.

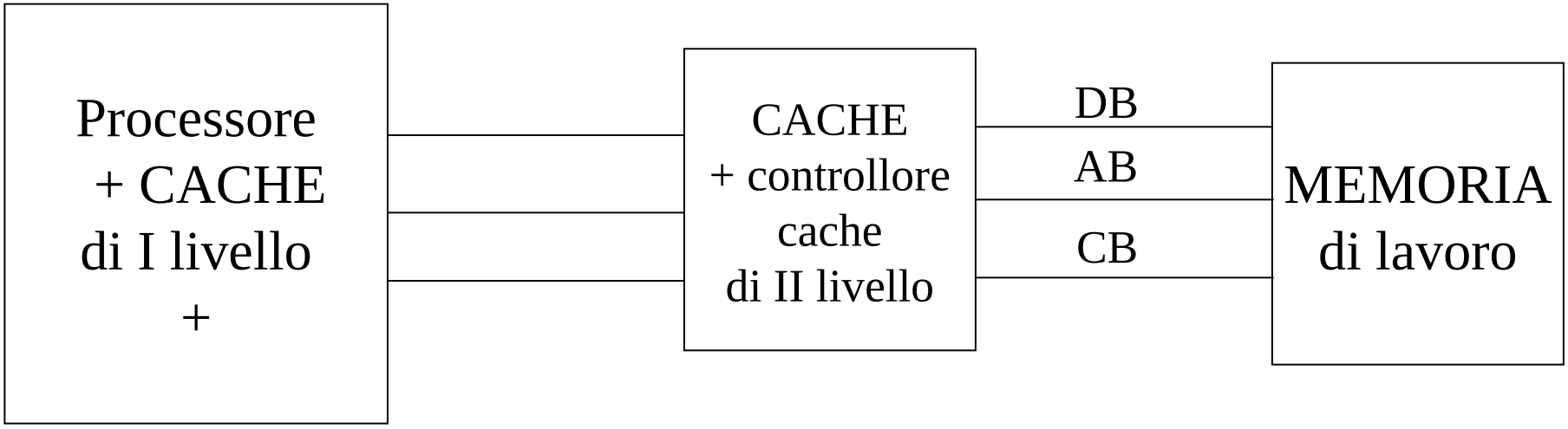
# INSERIMENTO DI MEMORIA CACHE DI SECONDO LIVELLO

Per il momento, per semplicità di esposizione, si ipotizza che

spazio di memoria di lavoro = spazio indirizzabile dal processore

Successivamente ipotizzeremo che tale spazio sia inferiore e che quindi sia necessario una memoria di massa allo stato solido o magnetica.

Inoltre ipotizzeremo, sempre per semplicità che anche la memoria cache di II livello sia strutturata come quella di I livello (ad accesso diretto) e con le stesse caratteristiche, ma di dimensioni maggiori. Lo schema di riferimento, quindi è quello indicato in Figura.

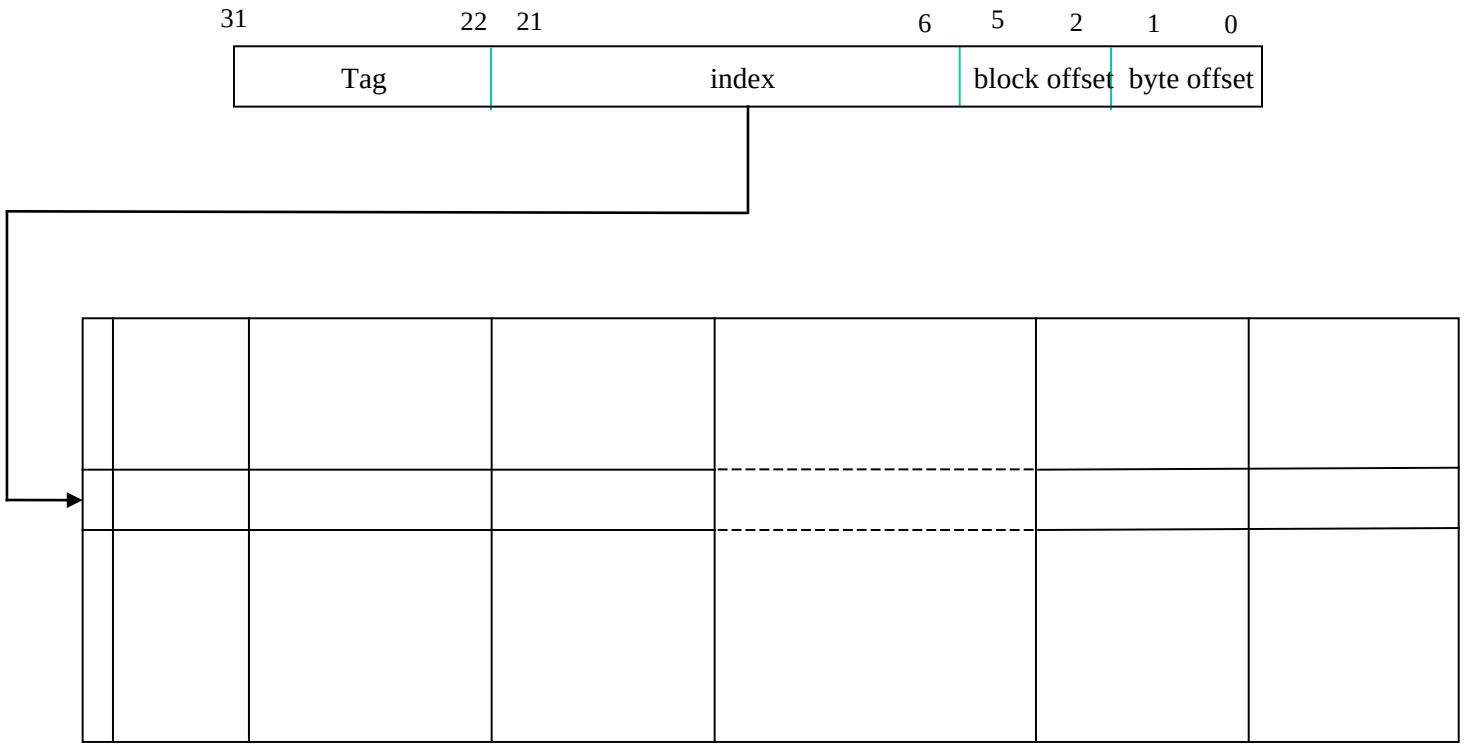


L’elemento inserito sarà un sistema digitale complesso strutturato con una SCA e uno SCO. E’ da notare, che nel caso di presenza solo della cache di I livello si era ipotizzato, per semplicità didattica, che la memoria di lavoro fosse statica, nel prosieguo continueremo ancora ad ipotizzare che la memoria di lavoro sia statica, come le due cache.

Il SCA del processore sarà identico a quello del caso si presenza di un solo livello di cache, e quindi il processore si interfaccerà verso la cache di II livello solo con il registro EMAR (Extended Memory Address Register) e EMDR (Extended Memory Data Register), mentre il SCO del processore, rispetto alla soluzione in cui si prevedeva un solo livello di cache, sarà leggermente diverso, in quanto l’interazione con la memoria di lavoro è delegato al controllore della cache di II livello e quindi nel caso di miss nel secondo livello, il processore dovrà attendere il trasferimento, in attesa attiva (campionando il segnale di ingresso WAIT negato), del blocco mancante dalla memoria di lavoro verso la cache di II livello, per poi interagire in lettura o scrittura con la cache di II livello, come fosse la memoria di lavoro del caso precedente (solo cache di I livello).

Di seguito si ipotizzerà di avere una cache di II livello ad accesso diretto caratterizzata da:
-un tag di 10 bit
-un index da 16 bit
-un block offset di 4 bit, e
-un byte offset di 2 bit

Pertanto si avrà una cache con $2^{16}$ righe, dove in ogni riga ci sono 16 dati, come schematizzato in Figura



# Interazione tra il I e il II livello

Di seguito si farà vedere come, dato un indirizzo di memoria, questo venga utilizzato per accedere ai dati nelle cache di I e II livello

p.e. dato l'indirizzo

01010100110100100010011110101 - - (dove gli ultimi d.c.c. sono dovuti eventualmente utilizzabili per accesso ad un byte)

Per la cache di I livello:

-i bit da $a_{31}$ a $a_{14}$ (i.e 01010100110100100 0): sono utilizzati per il campo TAG

-i bit da $a_{13}$ a $a_{6}$ (i.e. 10011111): sono utilizzati per il campo INDEX

-i bit da $a_{5}$ a $a_{2}$ (i.e. 0101): sono utilizzati per il campo BLOCK OFFSET

Mentre per la cache di II livello:

-i bit da $a_{31}$ a $a_{22}$ (i.e 0101010011): sono utilizzati per il campo TAG

-i bit da $a_{21}$ a $a_{6}$ (i.e 0100100010011111): sono utilizzati per il campo INDEX

-i bit da $a_{5}$ a $a_{2}$ (i.e 0101): sono utilizzati per il campo BLOCK OFFSET

Nella Figura successiva c'è la rappresentazione grafica di tale organizzazione

| TAG 2 | INDEX 2 | BLOCK OFF 2 |
| --- | --- | --- |
| $a_{31} \ldots a_{22}$ | $a_{21} \ldots a_{6}$ | $a_{5} \ldots a_{2}$ |

*01010100110100100010011110101 --*

*01010100110100100010011110101 --*

| TAG 1 | INDEX 1 | BLOCK OFF 1 |
| --- | --- | --- |
| $a_{31} \ldots a_{14}$ | $a_{13} \ldots a_{6}$ | $a_{5} \ldots a_{2}$ |

Di seguito si farà vedere il SCA necessario e il SCO del controllore della cache di II livello nel caso di:

- lettura dati dalla cache di II livello per scrivere sulla cache di I livello

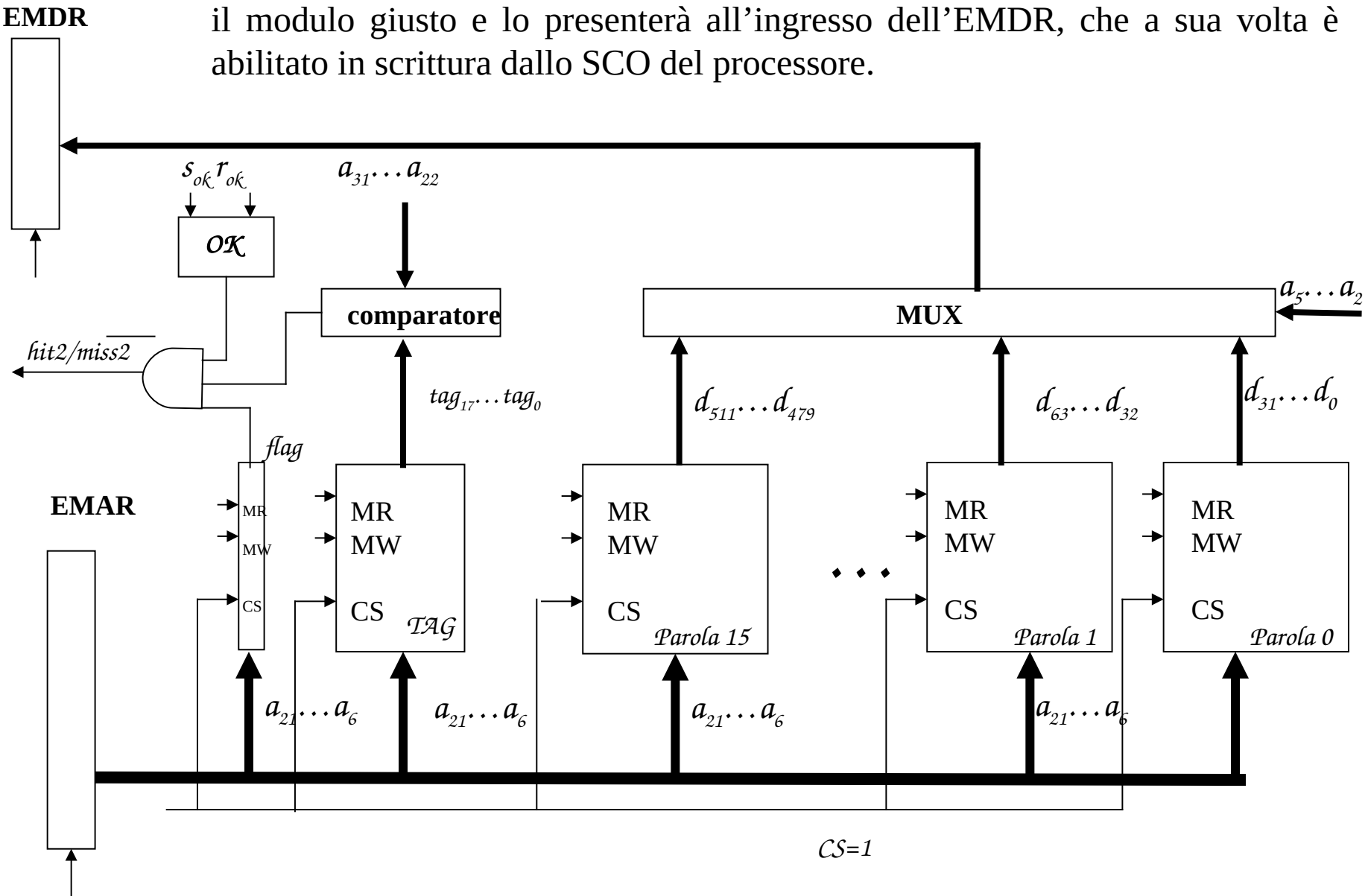
- scrittura dati sulla cache di II livello dalla cache di I livello

- scrittura dati sulla cache di II livello dalla memoria di lavoro

- lettura dati dalla cache di II livello per scrivere sulla memoria di lavoro

## SCHEMA SEMPLIFICATO LETTURA dalla CACHE di II livello

Nel caso di hit è lo SCO del processore che si «legge» il dato dalla cache di II livello generando il segnale di MR per tutti i moduli, il MUX che selezionerà il modulo giusto e lo presenterà all'ingresso dell'EMDR, che a sua volta è abilitato in scrittura dallo SCO del processore.



**SCHEMA SEMPLIFICATO LETTURA dalla CACHE di II livello**

Le connessioni disegnate servono solo per permettere il trasferimento dati dalla cache di II livello verso la cache di I livello.

Nel caso di hit si conclude il normale ciclo di lettura, quindi si introduce uno stato in cui una volta trasferito l'indirizzo a cui si vuole accedere si verifica se c'è hit o meno.

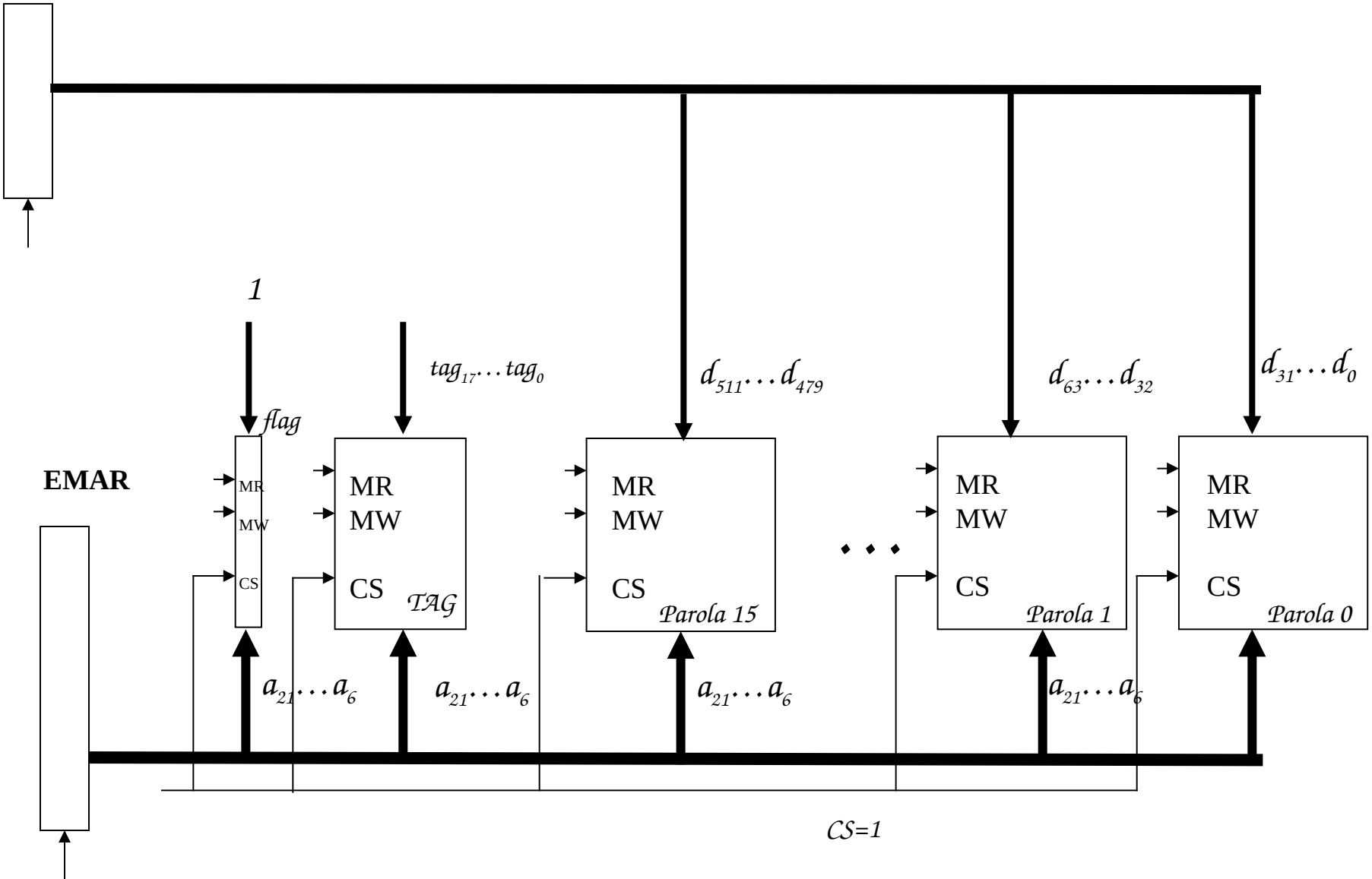
Nel caso di **miss** (hit/miss negato = 0) il segnale di controllo MR generato dallo SCO del processore non ha effetto sui moduli di memoria della cache di II livello in quanto tale segnale non lo si manda direttamente ai moduli ma dopo che gli si è fatto un AND con il segnale hit/miss (negato), pertanto quando c'è una miss il segnale di controllo posto all'ingresso dei moduli di memoria è pari a a ZERO, Tale segnale diventerà UNO dopo che lo SCO della cache di II livello avrà prelevato i dati dalla memoria e pertanto il segnale di controllo hit/miss (negato) diventerà UNO. Da ricordare che lo SCO della cache di II livello dovrà accedere alla memoria per prelevare i blocchi di dati da sostituire con quelli nella cache, però prima di sostituirli, scrive i dati dei blocchi presenti nella cache nella memoria di lavoro. Una volta trasferiti i blocchi dei dati dalla memoria alla cache si potrà concludere la lettura ( il segnale da miss diverrà hit).

Il F/F OK serve allo SCO del controllore della cache di II livello per lasciare il segnale di miss fino a che non è completamente completato il trasferimento dei 16 bocchi dalla memoria di lavoro alla cache stessa. Quindi in caso di miss metterà tale F/F a zero per rimetterlo ad uno a completamento del trasferimento.

Notare, vedere slides, successive, che nel modulo TAG in caso di scrittura verranno memorizzati i 10 bit più significativi dell'indirizzo.

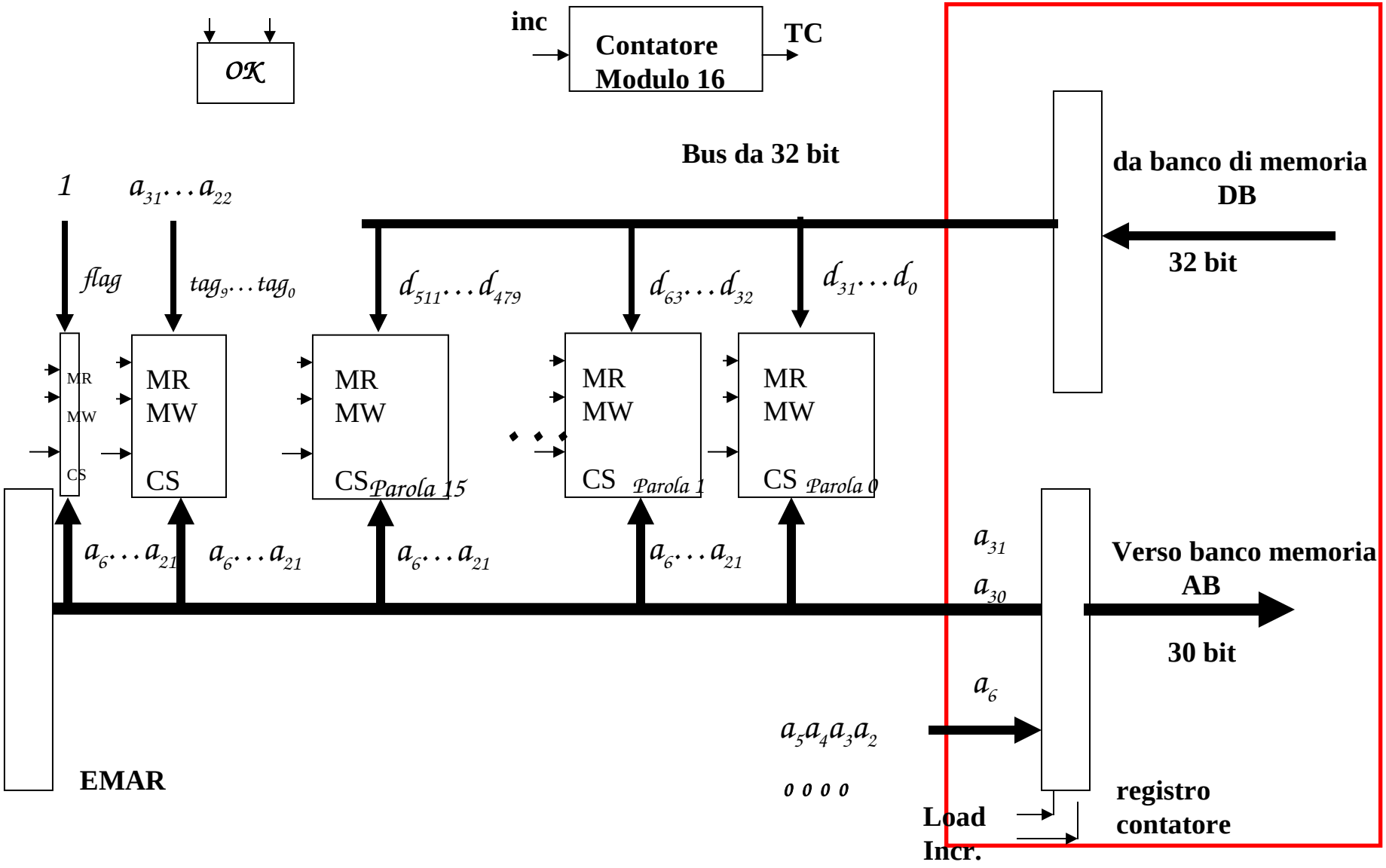
I bit del modulo di flag vengono messi tutti a zero in fase di accensione del sistema e vengono messi ad uno mano a mano che vengono caricate le long word prese dalla memoria di lavoro. Da notare che in caso di presenza del S.O. tali flag verrebbero posti a zero ogni qual volta ci fosse l'attivazione di un altro processo. POSSIAMO DIRE CHE LA MODALITà DI FUNZIONAMENTO CHE PREVEDIAMO è QUELLA DEL KERNEL, IN QUANTO TUTTI I DATI DEL KERNEL SONO NELLA MEMORIA DI LAVORO E QUINDI è COME SE LO SPAZIO DI INDIRIZZAMENTO DELLA MEMORIA FOSSE QUELLO DI DIMENSIONE MASSIMA ($2^{32}$ )

**SCHEMA SEMPLIFICATO di scrittura nella CACHE di II livello, dalla cache di I livello**



**SCHEMA SEMPLIFICATO SCRITTURA DATI NELLA CACHE II livello – LETTURA DALLA MEMORIA**

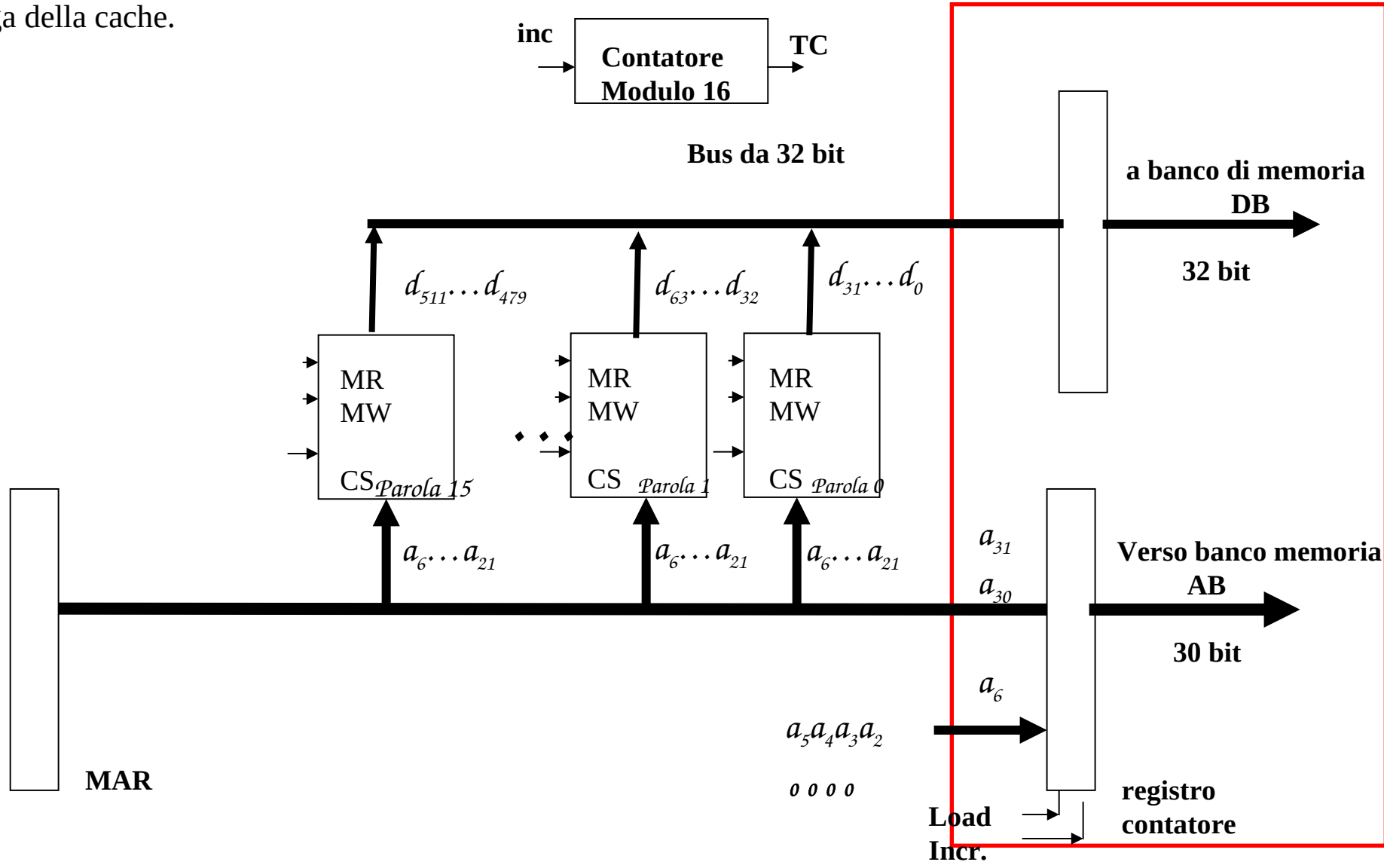
(è lo SCO della cache di II livello che gestisce la scrittura, con le temporizzazioni simili a quelle viste nel caso in cui era il processore che scriveva in memoria (caso in cui non c'è la cache di II livello))



# SCHEMA SEMPLIFICATO SCRITTURA DATI DALLA CACHE NELLA MEMORIA

Le connessioni disegnate servono solo per permettere il trasferimento dati dalla CACHE verso la MEMORIA, questa parte si attiva in caso di miss in lettura o in scrittura, IN QUANTO BISOGNA MEMORIZZARE I DATI DALLA CACHE VERSO LA MEMORIA PRIMA DI SOVRASCRIVERE.

Naturalmente se c'è una miss su una lettura/scrittura verranno copiate tutte e 16 le long word della stessa riga della cache.



**Superamento dell’ipotesi che lo spazio della memoria di lavoro è uguale a quello indirizzabile fisicamente dal processore**

Fino ad ora si è ipotizzato che lo spazio di indirizzamento della memoria fosse uguale a quello generabile dal processore, ma nel caso di 64 bit si avrebbe necessità di una memoria di lavoro di $2^{64}$ indirizzi, ma anche nel caso di 48 bit si avrebbe bisogno di una memoria di $2^{48}$ indirizzi, valori obiettivamente molto grandi e improponibili per il loro costo. Pertanto le memorie di lavoro sono normalmente di dimensioni inferiori rispetto allo spazio di indirizzamento del processore.

Inoltre si era ipotizzato che esistesse un unico programma con le relative strutture dati e con eventualmente tutti i driver relativi. In tal caso il programmatore potrebbe suddividere il programma e i dati in sottoinsiemi (normalmente di dimensioni uguali, che denoteremo come ***blocchi*** per non utilizzare il termine ***pagine*** tipico dei Sistemi Operativi), di cui solo alcuni in memoria di lavoro ed altri sul sistema di memorizzazione di massa (per esempio un disco magnetico o un dispositivo di realizzato con tecnologia flash) e progettare il programma in modo di accedere alla memoria di lavoro solo quando effettivamente il dato o l'istruzione da prelevare fosse effettivamente in memoria di lavoro. Invece, nel caso in cui dovesse avere necessità di dati o istruzioni memorizzate in blocchi memorizzati su disco allora il programmatore dovrebbe prima trasferire tali dati dal disco verso la memoria di lavoro, naturalmente, prima di poter far ciò sarebbe opportuno trasferire i contenuti del blocco precedente in memoria di lavoro su quella di massa. Questa modalità, conosciuta come ***overlay***, era effettivamente utilizzata agli albori dei sistemi di elaborazione, addirittura prima della introduzione dei dischi, quando i dati e le istruzioni venivano trasferite da schede perforate. Purtroppo tale modalità di programmazione è molto pesante e prona ad errori, per questo motivo è stato introdotto un meccanismo, la ***memoria virtuale***, che non solo alleggerisce il lavoro di programmazione del singolo programma, ma consente di poter eseguire contemporaneamente più programmi, ognuno con il proprio spazio di indirizzamento.

Come organizzare lo spazio di indirizzamento virtuale in quello reale e quali supporti logici e fisici sono necessari per l'implementazione della memoria virtuale è al di fuori dagli obiettivi del presente libro e oggetto dei corsi sui Sistemi Operativi, a cui si rimanda per maggiori dettagli. Comunque le attività che compie il Sistema Operativo sono analoghe a quelle che dovrebbe fare un programma assembly nel caso di assenza della memoria virtuale e cioè nell'accesso in memoria di lavoro (ipotizzando l'organizzazione del programma e dei dati in blocchi):

A.in caso di assenza del dato nella memoria di lavoro:

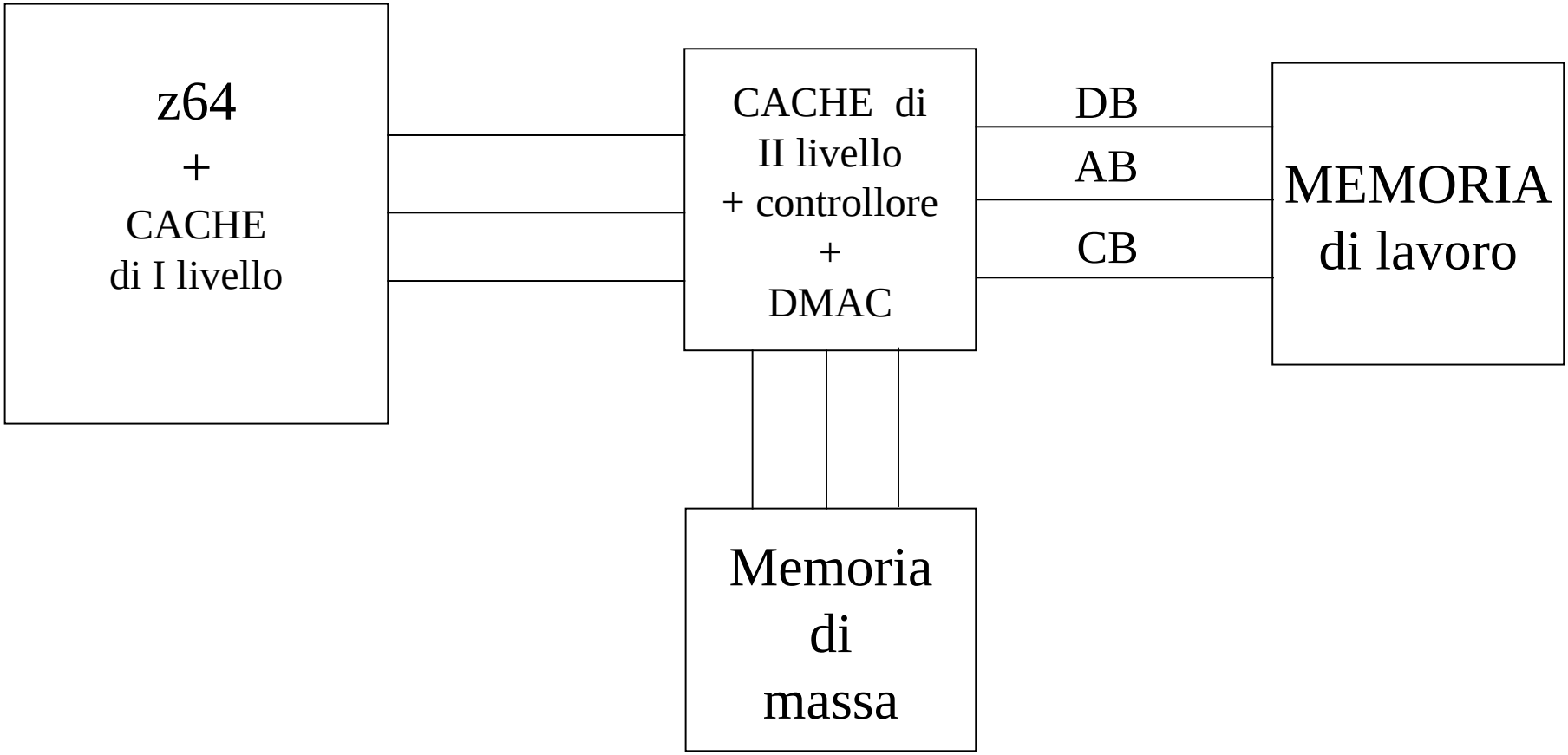
- verifica della disponibilità di spazio sufficiente nella memoria di lavoro per trasferire una pagina dalla memoria di massa
    1. in caso di mancanza di spazio sulla memoria di lavoro:
        - trasferire una o più pagine presenti sulla memoria di lavoro e loro trasferimento in memoria di massa
        - prelievo della pagina dalla memoria di massa e suo trasferimento in memoria di lavoro
    2. in caso di disponibilità di spazio sulla memoria di lavoro:
        - prelievo della pagina dalla memoria di massa e suo trasferimento in memoria di lavoro

B.in caso di presenza: accesso alla memoria di lavoro

Le attività elencate si basano su due tipi di attività elementari:
1.Lettura/scrittura del processore di informazioni da/verso la memoria
2.Trasferimento da/a memoria di lavoro a/da memoria di massa di un segmento

La prima attività è identica a quella che abbiamo visto nel caso in cui lo spazio di indirizzamento del processore è uguale a quella delle dimensioni della memoria di lavoro, mentre il modo più efficiente di traferire dati per la seconda attività è quella di utilizzare un DMAC, così come visto nel Capitolo???

Pertanto una soluzione in cui si possa accedere a informazioni che si possono trovare su cache di I livello, su cache di II livello, su memoria di lavoro o su memoria di massa può essere schematizzata come in Figura ??? (dove la cache di I livello è all'interno del processore e quella di II livello esterna ad esso)



Da notare che quando si trasferisce un blocco di informazioni (dati e/o istruzioni) dalla memoria di massa alla memoria di lavoro è necessario evitare che ci siano copie di informazioni inconsistenti tra la memoria di lavoro e le cache di I e II livello, per questo motivo nel caso di sostituzione di un blocco della memoria con uno della memoria di massa è necessario ripulire le memorie cache dalle informazioni che sono state scritte nella memoria di massa.

Il modo più semplice per ripulire le memorie cache di I e II livello è quello azzerare i flag di tutte le righe delle memorie cache e questo può essere ottenuto facilmente modificando il microcodice del processore e del controllore della cache di II livello. In particolar modo ipotizzeremo che ogni qual volta che il DMAC finisce di trasferire un blocco di informazioni dalla memoria di massa alla memoria di lavoro invia un segnale di controllo al controllore della memoria cache di II livello e al processore affinché azzerino tutti i flag di validità delle relative memorie cache. Per essere efficienti si dovrebbero azzerare solo le righe relative agli indirizzi del blocco trasferito, ma questo invalidamento selettivo, pur se fattibile, non è di semplice implementazione e quindi per semplicità ipotizzeremo che ci sia solo l'invalidamento di tutte le righe delle cache.
Lo SCO del processore una volta che gli arriva il segnale di azzeramento delle cache di I livello mette a zero tutti i flag della sua cache, così come il controllore della cache di II livello. Questo può essere ottenuto inserendo un segnale di reset che opera in parallelo su tutti i flip/flop della colonna flag.

# DMAC e memoria virtuale

- In sistemi con memoria virtuale, il DMAC deve trasferire usando indirizzi virtuali o indirizzi fisici?
  - Se usa indirizzi fisici, il trasferimento non può riguardare facilmente più di una pagina
    - Più pagine non corrispondono generalmente a locazioni sequenziali in memoria
  - Se usa indirizzi virtuali, li deve tradurre in indirizzi fisici
    - Il sistema operativo fornisce delle tabelle di traduzione quando inizia il trasferimento