ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Facoltà di Ingegneria

Corso di Laurea in INGEGNERIA INFORMATICA

Progetto di CALCOLATORI ELETTRONICI M

Progetto di una memoria cache per il processore DLX

Componenti Gruppo:

Andrea Grandi

Filippo Malaguti

Massimiliano Mattetti

Gabriele Morlini

Thomas Ricci

Anno Accademico 2009/2010

Indice

Introduzione

La necessità di introdurre cache per un processore deriva dal noto problema del collo di bottiglia rappresentato dall'accesso a dispositivi di memoria. Un processore durante il suo funzionamento tende ad accedere in scrittura o a reperire dati in lettura provenienti dalla memoria a valle e tale operazione richiede tipicamente diversi cicli di clock che costringono il processore (più veloce della memoria) ad attendere il dato. Ciò comporta l'introduzione di cicli di wait che ovviamente causano un peggioramento delle performace del processore, il quale attende che la memoria gli presenti il dato richiesto segnalato dal segnale di ready.

Per superare tale problema si utilizzano pertanto delle memorie cache, vicine al processore, di piccole dimensioni e molto veloci (tanto che possono avere tempi d'accesso simili a quelli dei registri interni al processore) da cui vengono reperiti i dati necessari all'esecuzione, consentendo in caso di HIT, ovvero nel caso in cui il dato si trovi in cache, di recuperarlo quasi senza ritardo.

Le cache si posizionano nella gerarchia delle memorie (insieme ai registri) tra i livelli più prossimi al processore e ciò comporta da un lato la rapidità nell'accesso e dall'altro le dimensioni limitate che fanno sì che una cache contenga un subset delle linee di memoria del dispositivo a valle (memoria o un livello superiore di cache se presente).

Pertanto quando si accede a una cache possono capitare due casi:

1. il dato si trova nella cache (HIT);

2. il dato non è presente e deve essere recuperato da un dispositivo a valle (MISS).

Ovviamente in caso di MISS si deve pagare un costo temporale per il reperimento del dato assente, detto miss penalty, dato dalla somma del tempo d'accesso al dispositivo a valle e dal tempo di trasferimento della linea col dato cercato.
Ciononnostante, è dimostrato che l'hit rate e quindi l'efficienza delle cache è tipicamente molto alta (oltre il 95%) grazie alla validità del Principio di Località spaziale e temporale, per il quale un programma in esecuzione tende ad eseguire temporalmente istruzioni eseguite di recente e ad accedere a dati acceduti di recente. Quindi sulla base di tali considerazioni, l'uso di cache contenenti le linee di memoria più recentemente accedute (working set) consente di migliorare notevolmente il tempo di reperimento dei dati necessari all'esecuzione, evitando quindi i ritardi che si avrebbero per ogni accesso diretto in memoria.
Abbiamo scelto questo progetto per approfondire le tematiche e le problematiche legate alla progettazione di un componente cache da affiancare al processore DLX visto a lezione.

1 Obiettivi del progetto

L'attività di progetto svolta si prefigge i seguenti obiettivi

1. **Realizzazione memoria cache:** progetto di un component VHDL che realizza il funzionamento di una memoria cache generica.
2. **Testbench del component:** progetto di una suite di test per il component.
3. **Integrazione con DLX:** modifica del progetto DLX per consentire l'integrazione del component realizzato con il processore

Capitolo 1

Caratteristiche della memoria cache

Si è scelto di progettare una cache di tipo set-associative, la cui schematizzazione è mostrata in Fig. 1.1. Questa tipologia di cache rappresenta un buon compromesso tra flessibilità e costo in termini di silicio.

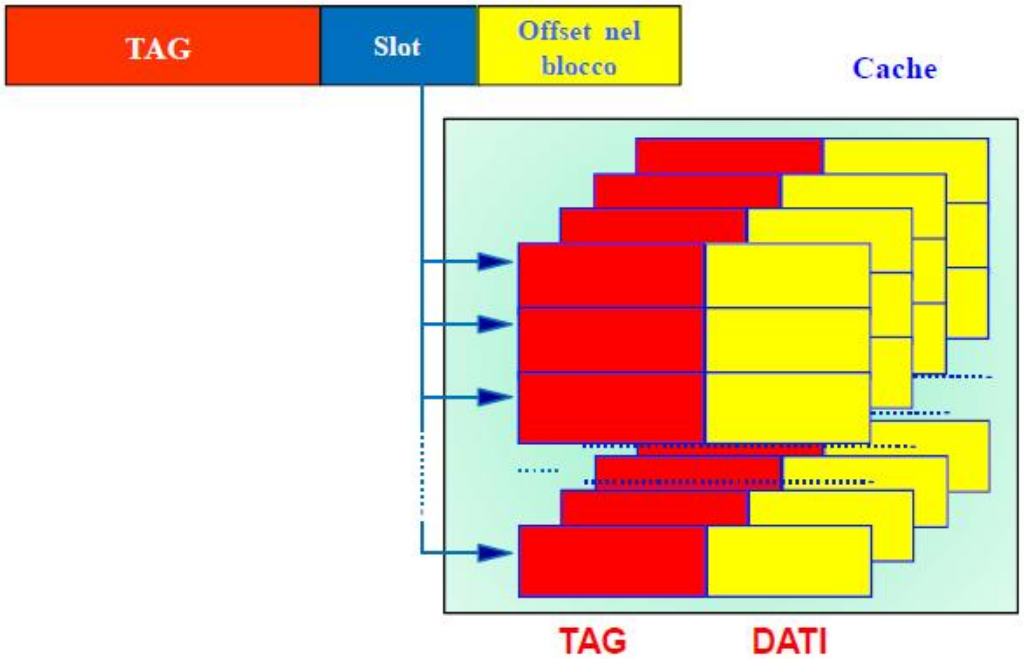

Figura 1.1: Schematizzazione di una cache set-associative

L'indirizzo di partenza del blocco è diviso in TAG (parte alta), INDEX e OFFSET (parte bassa). Il TAG consente di identificare univocamente una linea all'interno di un un sottoinsieme di linee, detto SET. L'INDEX individua immediatamente il SET all'interno del quale è possibile recupeare la linea corrente tramite il confronto del TAG. In questo modo si limita il numero di confronti tra TAG accettando il fatto che ogni linea possa appartenere ad un singolo set. La parte meno significativa dell'indirizzo rappresenta l'OFFSET che consente di individuare il dato all'interno di una linea.
Per garantire maggiore flessibilità si è scelto di parametrizzare alcune delle caratteristiche statiche della cache, quali ad esempio:

- la dimensione dei blocchi
- il numero di vie
- il numero di linee

1 Politica di rimpiazzamento

Nel caso in cui si debba caricare una nuova linea e tutte le vie siano occupate è necessario determinare quale linea rimpiazzare. Un buon algoritmo di rimpiazzamento dovrebbe cercare di individuare la linea vittima che meno probabilmente verrà riutilizzata in seguito. Il criterio scelto per effettuare il rimpiazzamento è basato su contatori, che implementa una politica LRU (Least Recently Used). Tale politica è tipicamente implementata poichè statisticamente si verifica principio di località. È quindi presente un contatore per ogni via di ogni set tramite il quale si tiene traccia di quanto recentemente si è acceduti a ciascuna linea: un valore basso del contatore indica un accesso recente mentre un valore alto indica un accesso*vetusto*. Evidentemente la linea candidata al rimpiazzamento risulta essere quella alla quale è associato il contatore di valore più elevato.
Nel caso di HIT su una linea, sono incrementati i valori di contatori

più basso rispetto al valore di quello della linea HIT mentre quest'ultimo viene resettato. Nel caso di MISS si procede con un rimpiazzamento e poi si agisce come nel caso di HIT sulla nuova linea. Infine, in caso di invalidazione di una linea, si porta al valore massimo il contatore della linea invalidata e si decrementano di 1 tutti i contatori con valore più elevato di quello della linea invalidata.

2 Struttura e interfacce

La memoria cache si interfaccia con i dispositivi esterni attraverso 4 tipi di interfacce, come mostrato in Fig. 1.2.

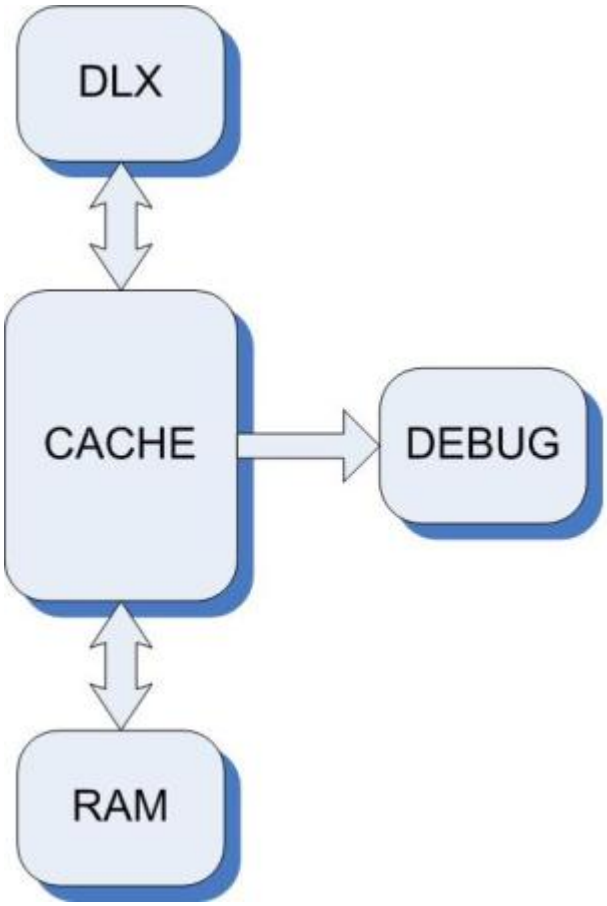

Figura 1.2: Interfacce della memoria cache

L'interfaccia verso il microprocessore, mostrata in Fig. 1.3, consente a quest'ultimo di accedere ai dati memorizzati all'interno della cache.

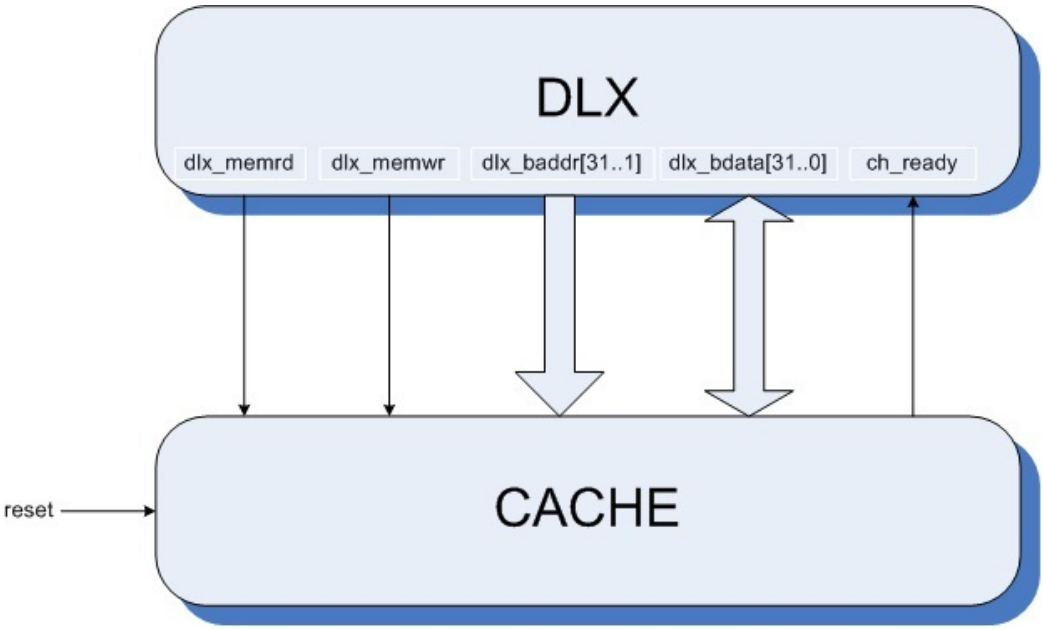

Figura 1.3: Interfaccia della memoria cache verso il processore DLX

In particolare sono presenti i seguenti segnali:

- **ch_baddr(31-2)**: indirizzi a 32 bit emessi dal microprocessore
- **ch_bdata(32-0)**: bus dati con parallelismo 32bit
- **ch_memwr**: segnale per il comando di scrittura in cache
- **ch_memrd**: segnale per il comando di lettura da cache
- **ch_ready**: segnale che indica il termine dell'operazione di lettura/scrittura corrente

Per quanto riguarda gli scambi di dati tra processore e memoria cache, si ipotizza che siano sempre lette e scritte parole di lunghezza fissa a 32 bit.

L'interfaccia verso il controllore di memoria, mostrata in Fig. 1.4, consente di testare e modificare lo stato delle linee.

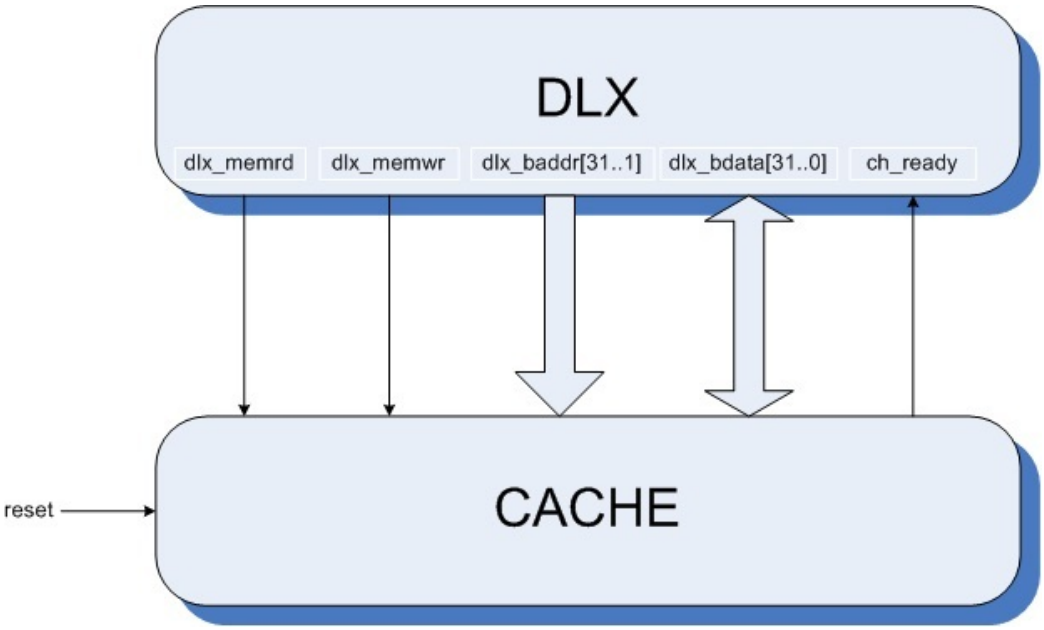

Figura 1.4: Interfaccia della memoria cache verso il controllore di memoria

In particolare sono presenti i seguenti segnali:

- **ch_eads**: inizia il ciclo di snoop
- **ch_inv**: invalida la linea
- **ch_hit**: indica che la linea richiesta è presente in memoria
- **ch_hitm**: indica che la linea richiesta è presente in memoria in stato modified
- **ch_flush**: scarica il contenuto della cache sulla RAM (tutto o solo un blocco?, funziona?)

Per semplificare la realizzazione del componente Cache, si è ipotizzato che la RAM fosse incapsulata all'interno dello stesso. In questo modo si riduce notevolmente il numero dei segnali esterni.

Una possibile interfaccia verso la RAM (non realmente implementata) è mostrata in Fig. 1.5 e consente alla cache di recuperare i blocchi dal livello sottostante.

In particolare sono presenti i seguenti segnali:

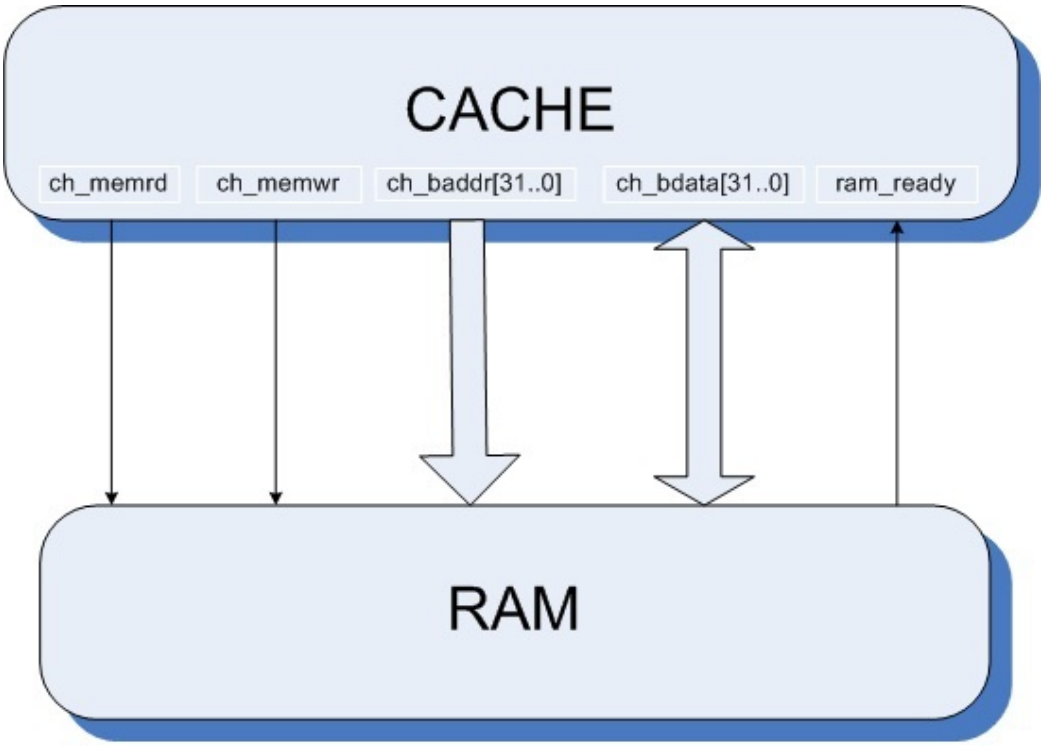

Figura 1.5: Interfaccia della memoria cache verso la RAM

- **Address(31-2)**: indirizzi a 32 bit emessi dalla cache
- **Data(32-0)**: bus dati con parallelismo 32bit
- **Write**: segnale per il comando di scrittura in RAM
- **Read**: segnale per il comando di lettura dalla RAM
- **Ready**: segnale che indica il termine dell'operazione di lettura/scrittura corrente

In realtà a livello di codice la RAM è integrata all'interno del componente Cache.
È presente infine una terza interfaccia verso l'esterno, utilizzata per monitorare lo stato interno della cache e poter quindi eseguire il debug. I segnali disponibili verranno definiti nel seguito.

Anche se la memoria cache progettata non verrà impiegata in sistemi multimaster, si è comunque deciso di affrontare alcune delle problematiche inerenti alla presenza di un controllore di memoria.

Tramite l'opportuna interfaccia è ad esempio possibile effettuare l'invalidazione delle linee e lo snooping dei dati presenti in cache.

Capitolo 2

Realizzazione e collaudo

La cache è stata realizzata come componente indipendente, detto `Cache_cmp`.
In questo capitolo saranno mostrate le caratteristiche principali di tale componente.

1 Strutture dati

Le strutture dati impiegate nel componente sono definite nel file `Cache_lib.vhd`.

Listing 2.1: Costanti e tipi di dato definiti nel file `Cache_lib.vhd`

```
CONSTANT OFFSET_BIT : natural := 5;
CONSTANT INDEX_BIT : natural := 2;
CONSTANT TAG_BIT : natural := PARALLELISM - INDEX_BIT -
    OFFSET_BIT;
CONSTANT NWAY : natural := 2;

CONSTANT MESI_M : natural := 3;
CONSTANT MESI_E : natural := 2;
CONSTANT MESI_S : natural := 1;
CONSTANT MESI_I : natural := 0;
```

```
TYPE data_line IS ARRAY (0 to 2**OFFSET_BIT - 1) of
    STD_LOGIC_VECTOR (7 downto 0);

TYPE cache_line IS
    RECORD
        data : data_line;
        status : natural;
        tag : STD_LOGIC_VECTOR (TAG_BIT-1 downto 0);
        lru_counter : natural;
    END RECORD;

TYPE set_ways IS ARRAY (0 to NWAY - 1) of cache_line;

TYPE cache_type IS ARRAY (natural range <>) of set_ways;
```

Il numero di bit di offset, indice e tag è stato parametrizzato per rendere più flessibile l'utilizzo del componente.
Sono stati inoltre definiti i seguenti tipi di dati:

- `data_line`: contiene i dati per una linea della cache, la cui dimensione è calcolata in base al numero di bit di offset;
- `cache_line`: record contenente le informazioni su dati e stato di una linea;
- `set_ways`: array di `NWAY` linee che compongono una via;
- `cache_type`: array di vie, costituisce l'intera cache ??? (non so come scrivere... :S).

Per ogni `cache_line` si tiene inoltre traccia di:

- `data`: `data_line` relativa alla linea corrente;
- `status`: indica lo stato MESI della linea;
- `tag`: bit dell'indirizzo che rappresentano il tag della linea;
- `lru_counter`: contatore usato dalla politica di rimpiazzamento.

2 Implementazione

All'interno del componente è presente un unico process detto `cache_process`, il quale risponde alle variazioni dei segnali di comando `ch_reset`, `ch_memrd`, `ch_memwr`, `ch_eads`.

All'interno del process vengono invocate le opportune procedure, tramite le quali si realizzano tutti i meccanismi per l'accesso e la modifica dei dati contenuti nella cache.

Listing 2.2: Codice VHDL del process `cache_process`

```
cache_process: process (ch_reset, ch_memrd, ch_memwr,
    ch_eads) is
    variable word: STD_LOGIC_VECTOR (31 downto 0):= (others
        => '0');
    variable hit: STD_LOGIC;
    variable hit_m: STD_LOGIC;
begin
    if (ch_reset = '1') then -- reset
        ch_ready <= '0';
        cache_reset;
        -- Inizializzazione cache e ram per il debug
        for i in 0 to 1023 loop
            RAM(i):= conv_std_logic_vector(i mod 256, 8);
        end loop;
    else
        if(ch_memrd = '1' and ch_memwr = '0') then -- memrd
            cache_read(word);
            ch_bdata_out <= word;
        elsif(ch_memrd = '0' and not ch_memwr'event and not
            ch_reset'event) then -- fine memrd
            ch_bdata_out <= (others => 'Z');
        end if;

        if(ch_memwr = '1' and ch_memrd = '0') then -- memwr
```

```
            word:= ch_bdata_in;
            cache_write(word);
        end if;

        if(ch_eads = '1') then -- snoop
            cache_snoop(hit, hit_m);
            ch_hit <= hit;
            ch_hitm <= hit_m;
        else
            ch_hit <= '0';
            ch_hitm <= '0';
        end if;

        ch_ready <= ch_memrd or ch_memwr;

    end if;

    ch_debug_cache <= cache;
end process cache_process;
```

Di seguito saranno brevemente descritte le procedure invocate all'interno del process.

2.1 cache_read

Parametri di output:

- `word`: dato letto

Descrizione:

1. Legge l'indirizzo dal bus separando index, tag e offset
2. Verifica se c'è un hit attraverso `get_way()`
3. In caso di MISS applica la politica di rimpiazzamento richiamando `cache_replace_line()`
4. Legge il dato dalla cache
5. Aggiorna i contatori attraverso `cache_hit_on()`
6. Pone il dato letto in `word` e lo restituisce

2.2 cache_replace_line

Parametri di output:

- selected_way: via sulla quale è stato caricato il dato rimpiazzato

Descrizione:

1. Individua la linea da rimpiazzare, cioè quella con `lru_counter` massimo
2. Controlla se la linea ha stato MESI_M e in tal caso ne fa il write-back invocando `ram_write()`
3. Carica il nuovo blocco nella cache sovrascrivendo il vecchio
4. Modifica il bit di stato in base al valore di WT_WB
5. Restituisce il numero della via sulla quale è presente il dato appena caricato

2.3 cache_hit_on

Parametri di input:

1. `hit_index`: indice al quale si è verificato l'hit
2. `hit_way`: via nella quale si è verificato l'hit

Descrizione:
Applica la politica di invecchiamento aggiornando i contatori, in particolare:

1. incrementa i contatori di valore più basso della via corrente specificata da `hit_way`
2. resetta il contatore della via corrente

2.4 cache_inv_on

Parametri di input:

- `inv_index`: indice da invalidare
- `inv_way`: via da invalidare

Descrizione:
Applica la politica di invecchiamento aggiornando i contatori, in particolare:

1. decrementa i contatori di valore più alto della via corrente specificata da `inv_way`
2. porta al valore massimo il contatore della via corrente

2.5 cache_write

Parametri di input:

- `word`: parola ad scrivere nella cache

Descrizione:

1. Legge l'indirizzo dal bus separando index, tag e offset.
2. Verifica se c'è un hit attraverso `get_way()`
3. In caso di MISS applica la politica di rimpiazzamento richiamando `cache_replace_line()`
4. Scrive il nuovo dato sulla cache
5. Aggiorna i contatori attraverso `cache_hit_on()`
6. Aggiorna il bit di stato ed esegue eventualmente il write-through.

2.6 get_way

Parametri di input:

1. `index`: indice
2. `tag`: tag da controllare

Parametri di output:

- `way`: via nella quale è presente il dato

Descrizione:

1. Verifica se il dato è in cache, cioè se esiste una linea con tag uguale a quello specificato il cui stato è diverso da `MESI_I`
2. Se il dato non è presente restituisce way = -1
3. Se il dato è presente restituisce il numero della via

2.7 ram_write

Parametri di input:

- `tag`: tag della linea da scrivere
- `index`: index della linea da scrivere
- `way`: numero di via in cui si trova la linea da scrivere

Descrizione: 1. Costruisce l'indirizzo del blocco a partire da `tag` e `index` 2. Scrive i dati contenuti nel blocco sulla RAM

2.8 snoop

Da dettagliare in seguito ????

3 Diagrammi temporali

4 Problematiche principali affrontate

(metteri anche tutti i problemi relativi al bus bidirezionale)

Capitolo 3

Integrazione con DLX

Schemi a blocchi
Diagrammi temporali

Capitolo 4

Block RAM

Il nostro progetto così non può funzionare ¿ introduzione a Block RAM.
Come funziona a grandi linee
Qualche parte di codice

Bibliografia