

Paginazione: complementi

G. Lettieri

28 Aprile 2022

Eliminiamo ora tutte le semplificazioni che abbiamo introdotto e studiamo la MMU che si trova nei sistemi Intel/AMD a 64 bit.

1 Memoria fisica in memoria virtuale

La MMU₁ aveva una memoria interna per memorizzare le tabelle dei vari livelli, ma per la vera MMU non è così. Le tabelle devono essere memorizzate nella memoria fisica, insieme alle pagine dei processi e al codice e alle altre strutture dati del sistema. Ogni volta che il sistema carica un processo dallo swap deve anche trovare lo spazio per le necessarie tabelle di corrispondenza. Ogni volta che il sistema mette in esecuzione un processo, deve scrivere in `cr3` il numero del frame che contiene la tabella di livello 4 di quel processo.

Concettualmente tutte le tabelle di traduzione farebbero parte di M1: sono strutture dati del sistema, inaccessibili agli utenti. Devono però essere allocate dinamicamente, in base ai processi che vengono creati dalle applicazioni degli utenti. Dal momento che tutte le tabelle sono grandi 4 KiB, come le pagine, conviene allocarle dentro i frame della memoria M2, in modo che tutto lo spazio M2 sia potenzialmente disponibile sia per le pagine, sia per le tabelle, senza dover stabilire *a-priori* quanto spazio dedicare alle une o alle altre. Allocandole nei frame di M2 rispettiamo anche automaticamente i requisiti di allineamento (ricordiamo che tutte le tabelle devono essere allineate naturalmente).

Questo comporta, però, che il sistema deve poter accedere liberamente a tutti i frame di M2, in modo da poter consultare liberamente le tabelle (per esempio, quando deve eseguire lo swap-out di un processo, per andare a ritrovare tutte le pagine). Il sistema deve anche, ovviamente, accedere a tutto M1, quindi dobbiamo permettergli di accedere a tutta la memoria fisica. Allo stesso tempo dobbiamo continuare a negare questo accesso ai processi utente, se non per le parti che contengono le loro pagine. La cosa più semplice sarebbe di avere una MMU che si disattiva ogni volta che la CPU passa a livello sistema, ma la MMU che abbiamo non si comporta così. Possiamo però creare una traduzione che non abbia alcun effetto, che di fatto è equivalente a disattivarla. Creiamo quindi delle traduzioni “identità” che lascino inalterati (li traducano in sé stessi) tutti gli indirizzi che vanno da 0 fino all’ultimo indirizzo della memoria fisica. Poi, inseriamo queste traduzioni nello spazio di indirizzamento di ogni processo.

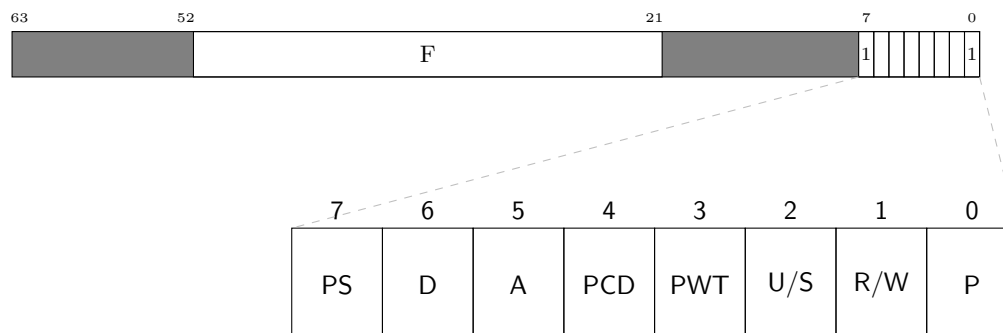


Figura 1: Descrittore di pagina virtuale da 2 MiB (tabelle di livello 2).

Questo permette alle routine di sistema di usare indirizzi fisici proprio come se la MMU fosse disattivata, indipendentemente da quale processo si trova in esecuzione. È come se nella parte alta dello spazio di indirizzamento di ogni processo avessimo creato una finestra che permette di vedere la memoria fisica così come è. Ovviamente questa finestra è accessibile solo da livello sistema, in quanto per tutte le traduzioni nella metà alta dello spazio di indirizzamento abbiamo posto U/S=sistema.

Questa finestra deve essere creata prima di attivare la memoria virtuale. All'avvio del sistema, il processore parte con la memoria virtuale disattivata ed esegue la routine di inizializzazione. Questa può allocare e inizializzare tutte le tabelle necessarie alla definizione della finestra e poi attivare la memoria virtuale. A questo punto la routine di inizializzazione può continuare ad utilizzare indirizzi fisici come stava facendo prima dell'attivazione. Quando si passa a livello utente queste traduzioni diventano inaccessibili e ridiventano accessibili ogni volta che si ritorna a livello sistema.

Le rimanenti caratteristiche della MMU ci permettono di ridurre l'occupazione di spazio e il tempo richiesto per effettuare le traduzioni.

2 Pagine di grandi dimensioni

L'architettura Intel/AMD a 64 bit ci permette di avere pagine più grandi di 4 KiB usando il bit PS nei descrittori di livello 3 e 2. Anche questo bit è scritto dalle routine di sistema e è soltanto letto dalla MMU. La Fig. 1 mostra il formato del descrittore di livello 2 nel caso in cui il bit PS valga 1. Si vede che il descrittore contiene ora un campo F che va dai bit 21 a 51, invece del puntatore alla tabella di livello 1. La Fig. 2 mostra la traduzione da indirizzo virtuale a fisico eseguita in questo caso: quando la MMU arriva alla tabella di livello 2 e trova il bit PS a 1, usa come offset tutti i bit da 0 a 20 e li concatena al campo F trovato nel descrittore. In questo modo abbiamo eliminato una tabella di livello 1, risparmiando il relativo spazio. Il meccanismo è compatibile e può convivere con le pagine di 4 KiB: la MMU si comporta come sempre nei

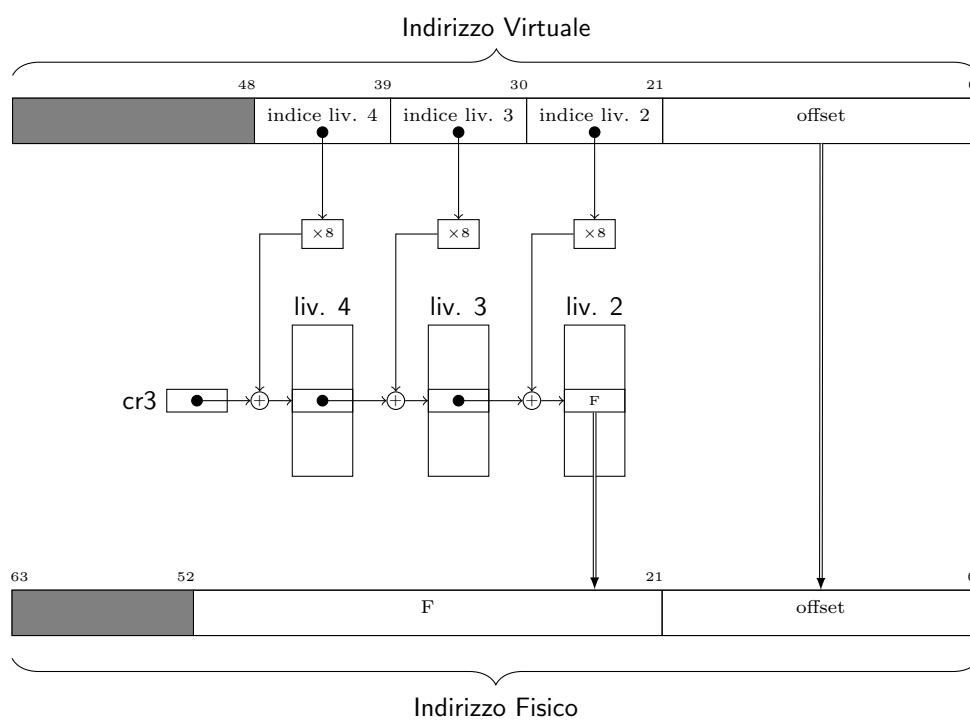


Figura 2: Traduzione da indirizzo virtuale a fisico (pagine di 2 MiB).

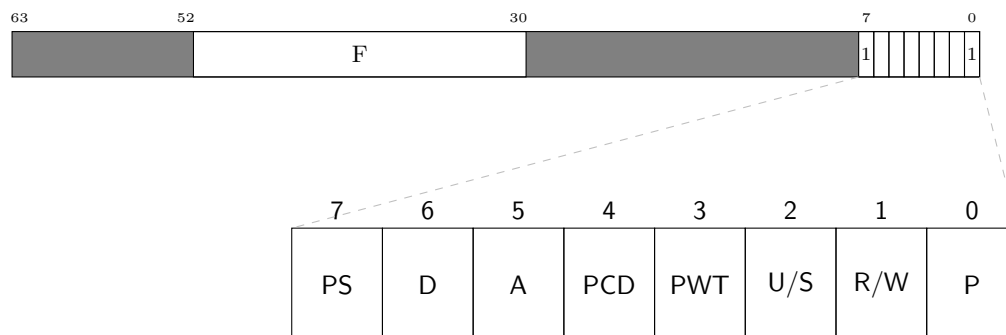


Figura 3: Descrittore di pagina virtuale da 1 GiB (tabelle di livello 3).

livelli 4 e 3 e scopre di dover eseguire il nuovo tipo di traduzione solo quando arriva al livello 2 e trova il bit PS pari a 1; se lo avesse trovato a 0 avrebbe proseguito fino al livello 1, come sempre. Ogni descrittore di livello 2 può avere il bit PS a 1 o a 0 indipendentemente dagli altri, quindi nello stesso spazio di indirizzamento possiamo usare sia pagine di 2 MiB, sia pagine di 4 KiB, a seconda della convenienza.

I processori più recenti permettono di avere PS=1 anche nei descrittori di livello 3. La Fig. 3 mostra il formato del descrittore di livello 3 in questo caso, in cui si vede che il campo F sostituisce anche qui il campo che punta alla tabella di livello 2. La Fig. 4 mostra la traduzione eseguita dalla MMU in questo caso. Si vede come l'offset è ora su 30 bit, e dunque le pagine sono ora grandi 1 GiB. Questo tipo di traduzione è molto utile per creare la finestra sulla memoria fisica (si veda la sezione precedente).

3 Il TLB

Per ogni accesso in memoria la MMU deve prima consultare fino a 4 tabelle per poter eseguire la traduzione¹. Per ogni tabella consultata, se il corrispondente bit A è a zero, la MMU deve anche eseguire un ulteriore accesso (in scrittura) per settarlo a 1. Se consideriamo che il nostro programma deve accedere continuamente in memoria, sia per prelevare le sue stesse istruzioni, sia per prelevare o scrivere gli operandi che si trovano in memoria, ci rendiamo subito conto che l'impatto della MMU sul tempo di esecuzione di un programma può essere molto grande. È vero che subito dopo la MMU c'è la cache, e quindi possiamo sperare che molti di questi accessi non debbano realmente arrivare fino alla memoria, ma resta il fatto che, anche nel migliore dei casi, quello che prima era un unico accesso in cache si è ora trasformato in una sequenza di 5 accessi in cache.

Per affrontare questo problema si introduce una nuova cache, chiamata TLB (Translation Lookaside Buffer), che è specifica per la MMU. Lo scopo di questa

¹Fino a 5 nei modelli con 57 bit significativi nell'indirizzo virtuale.

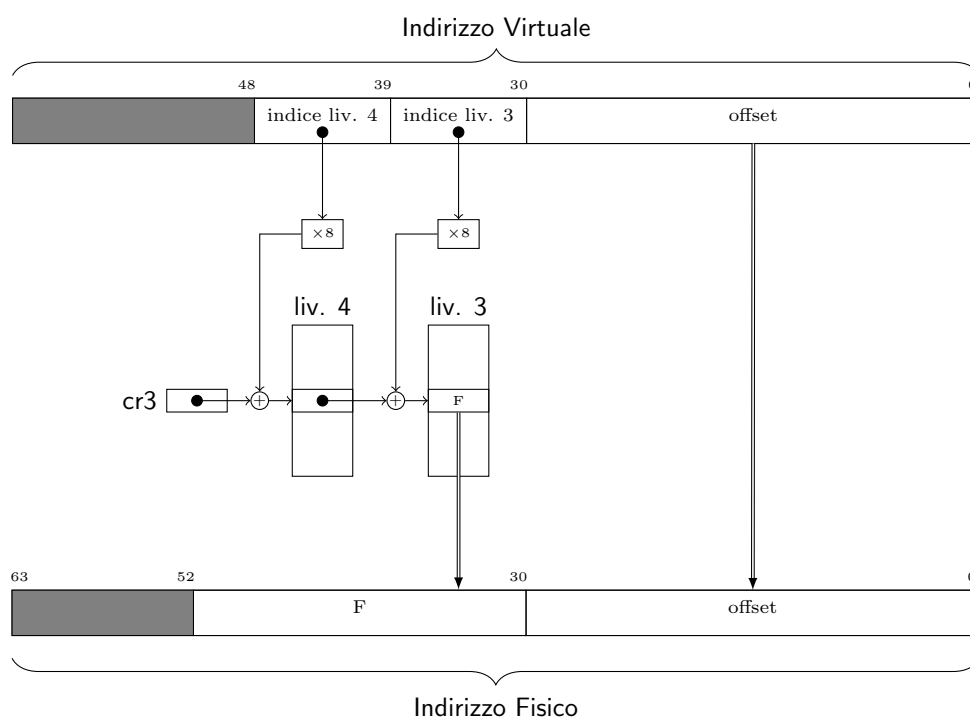


Figura 4: Traduzione da indirizzo virtuale a fisico (pagine di 1 GiB).

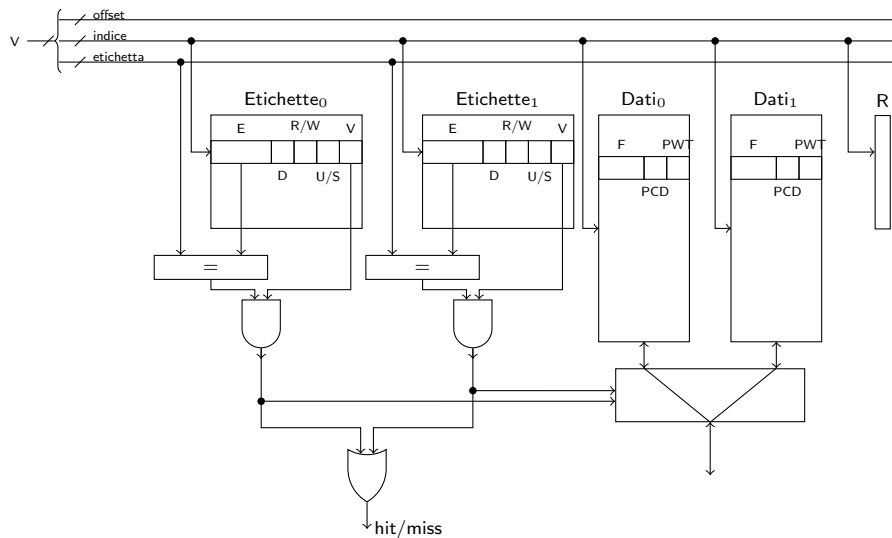


Figura 5: Un esempio di TLB a 2 vie.

cache è di ricordare le *traduzioni* utilizzate più recentemente, dove per traduzioni intendiamo, per le pagine di 4 KiB, quelle definite dai descrittori di livello 1, per le pagine di 2 MiB quelle nei descrittori di livello 2 e, infine, quelle dei descrittori di livello 3 per le pagine da 1 GiB. In ogni caso, per ognuna di queste traduzioni, non ci interessano i descrittori di livello più alto, il cui unico scopo è di permettere di raggiungere, dato l'indirizzo virtuale da tradurre, il descrittore che si trova in fondo alla catena. Una volta raggiunto questo descrittore la MMU può ricordare (nel TLB) quale sia la traduzione da fare per l'indirizzo virtuale in questione: non ha bisogno di ricordare tutto il percorso.

Consideriamo, per il momento, le sole pagine di 4 KiB. In questo caso il TLB è una cache dei descrittori di livello 1. Quando la MMU deve tradurre un indirizzo controlla prima se il TLB contiene già il descrittore che sta cercando; altrimenti si comporta come abbiamo visto fin'ora, seguendo tutta la catena di tabelle dal livello 4 fino al livello 1; alla fine, oltre ad eseguire la traduzione e completare l'accesso in memoria per conto del processore, memorizzerà nel TLB il descrittore appena usato, possibilmente rimpiazzandone un altro. Tipicamente il TLB è una memoria associativa ad insiemi e il descrittore da rimpiazzare sarà scelto in base ad un algoritmo di pseudo-LRU. In Figura 5 è mostrato un esempio di TLB a 2 vie. I campi dati di ogni via memorizzano i campi F dei descrittori di livello 1. La parte offset dell'indirizzo V in ingresso non è utilizzata per accedere alla memoria cache, ma andrà direttamente a far parte dell'indirizzo fisico. Un TLB moderno potrebbe avere 8 vie e contenere 1024 descrittori in totale (quindi $1024/8 = 128$ indici).

Il TLB, come tutte le cache, è trasparente al software, persino al software di sistema: l'architettura non prevede istruzioni che permettano al software di

esaminarne il contenuto. Le uniche istruzioni che l'architettura mette a disposizione sono quelle che permettono di *invalidarlo* in tutto o in parte. Queste operazioni si rendono necessarie quando il software modifica qualcosa nelle tabelle, rendendo dunque non più valide le informazioni che potrebbero trovarsi nel TLB. In particolare, le istruzioni disponibili per l'invalidazione sono due:

- `movq %rax, %cr3`, che già conosciamo; questa istruzione cambia potenzialmente tutta la tabella di livello 4 usata fino al momento prima, quindi tutte le informazioni contenute nel TLB sono da considerarsi non più valide e l'intero TLB viene svuotato;
- `invlpg operando in memoria`: questa istruzione dice al TLB di invalidare la traduzione relativa all'indirizzo dell'operando passato come argomento.

Ogni volta che si cambia processo verrà eseguita una `movq %rax, %cr3` per caricare il puntatore alla tabella di livello 4 del processo entrante. Questa istruzione avrà l'effetto di svuotare il TLB, che è quello che vogliamo, in quanto le traduzioni in esso presenti facevano riferimento alla memoria virtuale del processo uscente.

Il TLB deve permettere alla MMU di svolgere tutte le sue funzioni senza consultare l'albero di traduzione. Ricordiamo che la MMU, oltre ad eseguire la traduzione, deve anche controllare i permessi sui bit U/S e R/W, aggiornare i bit A e D e passare al controllore cache i bit PWT e PCD. Questi ultimi due bit vengono memorizzati insieme al numero di frame e, in caso di hit, passati alla MMU che li girerà al controllore cache.

Consideriamo i bit U/S: la MMU ne incontra fino a quattro durante la traduzione, ma non c'è bisogno che il TLB li ricordi tutti e quattro, in quanto la regola dice che l'accesso è vietato da livello utente se anche uno solo di essi è zero. È dunque sufficiente che la MMU calcoli l'AND dei bit incontrati nel percorso e carichi soltanto questo nel TLB. Lo stesso discorso vale per i bit R/W.

Un discorso diverso va fatto per i bit A e D, in quanto questi bit sono *scritti* dalla MMU durante la traduzione, e non vogliamo certo che la MMU sia costretta a percorrere l'albero per aggiornare questi bit anche quando ha trovato la traduzione nel TLB (cosa che renderebbe il TLB del tutto inutile). Per il bit A la soluzione adottata è semplice: la MMU li aggiorna solo quando non trova la traduzione nel TLB. Fino a che una traduzione si trova nel TLB, i corrispondenti bit A nel percorso di traduzione non verranno ulteriormente modificati dalla MMU. Questo è un problema solo se il software, per qualche motivo, li azzerava, perché in quel caso resterebbero a zero anche in presenza di ulteriori accessi. In questo caso deve essere il software stesso che si deve preoccupare di invalidare, in tutto o in parte, il TLB dopo aver azzerato i bit A, costringendo dunque la MMU a ripercorrere l'albero e ri-aggiornare i bit A al prossimo accesso.

Per il bit D il discorso è più complesso, perché può succedere che vi sia un primo accesso in sola lettura all'interno di una pagina, seguito da un accesso in scrittura. Nel primo accesso la MMU non porta, ovviamente, D a 1, e la

traduzione viene caricata nel TLB. Al secondo accesso la MMU trova la traduzione nel TLB e, se non prendessimo provvedimenti, non accedrebbe all'albero e lascerebbe il bit D sempre a zero, pur essendovi stata una scrittura all'interno della pagina. L'informazione offerta dal bit D sarebbe quindi inaffidabile. In questo caso è complicato trovare un rimedio puramente software, in quanto il software non sa cosa contenga il TLB. La soluzione adottata è hardware: il TLB ricorda anche il valore del bit D al momento del caricamento della traduzione e causa una *miss* per gli accessi in scrittura con D uguale a 0. Nell'esempio precedente accade dunque questo: nel primo accesso (sola lettura) il TLB carica la traduzione e il valore corrente di D, che è 0. Nel successivo accesso in scrittura il TLB si accorge che, anche se la traduzione è presente, il bit D è ancora a zero. Genera dunque un miss, forzando la MMU a percorrere l'albero, aggiornando di conseguenza il bit D. Si noti che a quel punto la MMU ricaricherà la traduzione nel TLB, questa volta con il bit D a 1: i successivi accessi in scrittura non causeranno dunque ulteriori miss.

Pensiamo ora alle pagine di grandi dimensioni, per esempio le pagine di 2 MiB. Il TLB per le pagine di 4 KiB non è in grado di memorizzare direttamente la traduzione di una pagina più grande, in quanto il numero di bit di offset è diverso. Se vogliamo continuare a usare un unico TLB, la traduzione di una pagina di 2 MiB deve essere scomposta nelle equivalenti 512 traduzioni di pagine da 4 KiB, occupando dunque 512 posizioni del TLB. Anche se alcuni processori più vecchi adottano questa soluzione, i processori moderni hanno invece un diverso TLB per ogni dimensione di pagina supportata. Si noti che la MMU deve cercare la traduzione in *ciascuno* di questi TLB, perché ogni indirizzo potrebbe essere tradotto usando pagine di qualunque dimensione, e non è possibile saperlo in anticipo. Per fortuna, la ricerca può essere svolta in parallelo, e dunque non influisce negativamente sulle prestazioni. La presenza di questi ulteriori TLB è un motivo in più per usare pagine di grandi dimensioni, quando possibile, in modo da alleggerire il TLB principale.