Calcolatori Elettronici: la memoria centrale

G. Lettieri

7 Marzo 2017

Ipotesi fondamentali da ricordare nel seguito:

- la memoria è organizzata in celle, ciascuna della capacità di un byte;
- una volta collegata al bus, ogni cella ha un proprio indirizzo, potenzialmente da 0 fino ad un massimo di $2^{64} - 1$[1];
- la memoria è in grado di eseguire soltanto operazioni di lettura o scrittura, una per volta;
- ciascuna operazione deve specificare l'indirizzo a cui si riferisce (da dove leggere o dove scrivere).

Il byte rappresenta l'unità di indirizzamento: non è possibile leggere o scrivere una unità più piccola di un byte in una singola operazione di lettura o scrittura. Se, per esempio, è necessario modificare un singolo bit all'interno di un byte, è necessario leggere l'intero byte, modificare il bit lasciando gli altri inalterati, quindi riscrivere il nuovo byte.

La memoria può anche leggere o scrivere più di un byte in una singola operazione e nello stesso tempo impiegato per un singolo byte. Nel nostro caso la memoria può leggere o scrivere una intera "parola quadrupla" di 64 bit (8 byte contigui) purché questa sia *allineata naturalmente*. La memoria che consideriamo può anche leggere o scrivere, in una sola operazione, una parola doppia di 32 bit (4 byte) o una parola di 16 bit (2 byte), purché queste non attraversino i confini di 8 byte (cioè siano interamente contenute in una regione di 8 byte).

Conviene dunque rappresentare la memoria non come una sequenza di byte, ma come una sequenza di regioni di 8 byte (il numero massimo di byte che può essere trasferito in una singola operazione), che chiamiamo *righe*. In pratica adottiamo una scomposizione degli indirizzi con $b = 3$. Si veda la Fig. 1, dove le righe sono disposte in orizzontale. In questo modo i dati accessibili in un'unica operazione saranno sempre contenuti in una singola riga.

Si noti che per avere le parole e le parole doppie sempre contenute in una riga è sufficiente che anch'esse siano allineate naturalmente (dunque a 2 le parole e a 4 le doppie), anche se non è necessario.

[1] $2^{52} - 1$ nell'architettura AMD64

numero di riga	+7	+6	+5	+4	+3	+2	+1	+0	indirizzo di riga
0									0
1									8
2									16
3									24
				...					
$2^{61}-1$									$2^{64}-8$

Figura 1: Organizzazione della memoria su righe di 64 bit. Ogni casella rappresenta un byte. I numeri +0, +1, ... sono gli offset all'interno della riga.

Per specificare completamente una richiesta di operazione di lettura dobbiamo presentare alla memoria l'indirizzo del primo byte e il numero di byte (per una operazione di scrittura dovremo anche presentare il nuovo contenuto dei byte in questione). Queste due informazioni vengono specificate in modo indiretto nel seguente modo:

1. scomponiamo l'indirizzo del primo byte in numero di riga e offset all'interno della riga;
2. prevediamo $64-3=61$ fili per il numero di riga, che chiamiamo A63–A3[2];
3. al posto dell'offset del primo byte e del numero di byte prevediamo 8 fili di *byte enable*, uno per ogni byte della riga.

Lo scopo delle linee di byte enable, che chiamiamo /BE7–/BE0, è di selezionare singolarmente i byte della riga che si intende leggere (o scrivere) nell'operazione.

Esempi:

- Lettura di una parola quadrupla all'indirizzo 512. Si tratta della riga n. $512/8=64$, che va dagli indirizzi 512 a 519. Avremo $\text{A63}=\text{A62}=\cdots=\text{A10}=0$, $\text{A9}=1$, $\text{A8}=\text{A7}=\cdots=\text{A3}=0$, in quanto 512 è 1000000000 in binario, e $\text{/BE7}=\text{/BE6}=\cdots=\text{/BE0}=0$ (tutti i byte abilitati);
- Lettura di una parola doppia all'indirizzo 512: A63–A3 come prima, ma $\text{/BE7}=\text{/BE6}=\text{/BE5}=\text{/BE4}=1$ (byte 7–4 disabiliati) e $\text{/BE3}=\text{/BE2}=\text{/BE1}=\text{/BE0}=0$ (byte 3–0 abilitati);
- Lettura di una parola doppia all'indirizzo 516: A63–A3 ancora come prima, in quanto siamo sempre all'interno della stessa riga. Avremo $\text{/BE7}=\text{/BE6}=\text{/BE5}=\text{/BE4}=0$ (byte 7–4 abilitati) e $\text{/BE3}=\text{/BE2}=\text{/BE1}=\text{/BE0}=1$ (byte 3–0 disabilitati).

Si noti che questo tipo di codifica permetterebbe di specificare molti più casi di quelli che ci servono. Per esempio, con $\text{/BE7}=\text{/BE5}=\text{/BE3}=\text{/BE1}=1$ e $\text{/BE6}=\text{/BE4}=\text{/BE2}=\text{/BE0}=0$ potremmo leggere solo i byte di indirizzo

[2] A52–A3 per AMD64.

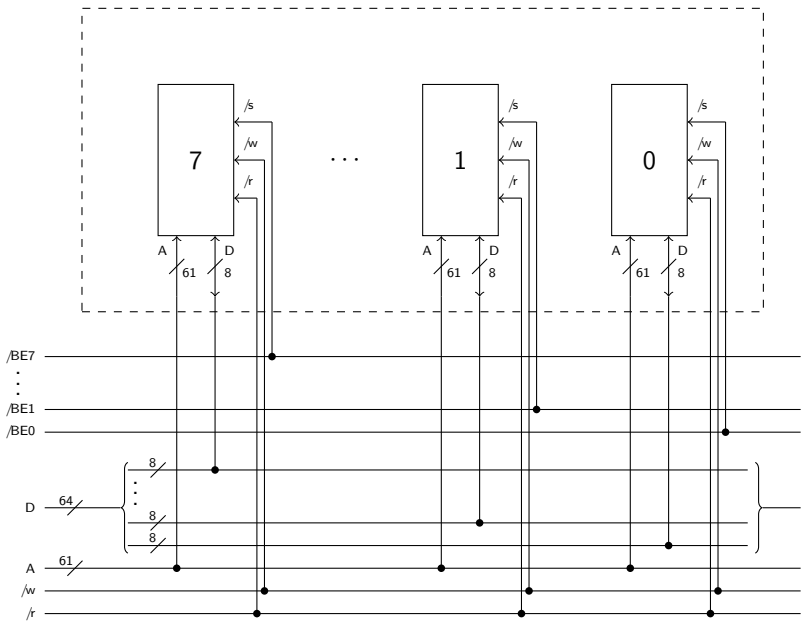

Figura 2: Realizzazione di una memoria organizzata a linee di 64 bit (parole quadruple), ma accessibile anche a byte, parole e doppie parole. Le parti dentro e fuori del rettangolo tratteggiato possono essere realizzate separatemente e poi collegate. La parte interna è una scheda di memoria, la parte esterna è il bus.

pari all'interno della riga selezionata. Di fatto tali possibilità non sono sfruttate e sono utilizzate solo le combinazioni che corrispondono a byte contigui.

Dal momento che lavoriamo con parole di più byte, ci possiamo chiedere in che ordine i byte della parola si trovano in memoria. Architetture diverse possono utilizzare ordini diversi. Nel nostro caso, l'architettura usa l'ordinamento detto *little endian*: il byte meno significativo si trova all'indirizzo più piccolo, seguito dagli altri byte in ordine di significatività. Per esempio, supponiamo di avere una parola quadrupla in memoria, a partire da un indirizzo i, che memorizza il numero esadecimale 0x1122334455667788. Il byte di indirizzo i conterrà 0x88, il byte di indirizzo $i + 1$ conterrà 0x77, e così via fino al byte di indirizzo $i + 7$ che conterrà 0x11. Un altro ordinamento molto utilizzato (per esempio da noi stessi quando scriviamo i numeri su un foglio) è quello *big endian*, che consiste nello scrivere il numero partendo dalla parte più significativa. In Fig. 1 abbiamo ordinato i byte di ogni riga da destra verso sinistra proprio per ovviare alla differenza tra il modo in cui l'architettura AMD64 memorizza le parole e il modo in cui noi siamo abituati a leggerle.

Una memoria con le caratteristiche che abbiamo descritto può essere realizzata supponendo di avere 8 dispositivi di memoria, ciascuno capace di memorizzare 2^{61} byte, collegati come in Fig. 2. Ciascuna riga di Fig. 1 è memorizzata uti-

lizzando tutti e 8 i dispositivi: il byte della colonna "+0" sarà memorizzato nel dispositivo contrassegnato con 0, il byte "+1" nel dispositivo 1, e così via. Ogni dispositivo memorizza una intera colonna di Fig. 1. Si noti come la presenza dei byte enable semplifichi l'implementazione: ogni byte enable è direttamente collegato al *chip select* del chip che contiene il corrispondente byte.

La memoria di Fig. 2 non è realistica nelle dimensioni. Una memoria realizzabile praticamente sarà molto più piccola, anche se gli indirizzi di riga potenzialmente generabili sul bus continuano ad essere 2^{61}. Consideriamo una memoria da 2^k byte, per esempio $k = 30$ per una memoria di 1 GiB. Quello che possiamo fare è di scegliere una regione di 1 GiB e fare in modo che la nostra memoria la occupi interamente. La nostra memoria dovrà rispondere alle richieste con indirizzi che cadono all'iterno della regione scelta, e ignorare tutte le altre (è importante che le ignori, per evitare conflitti con eventuale altra memoria che potremmo voler aggiungere in seguito, oppure con interfacce i cui registri sono mappati in memoria).

Sia r il numero della regione che abbiamo scelto. Data una operazione di lettura o scrittura ad un certo numero di riga i, la nostra memoria deve sapere se ignorarla o no in base al fatto che il numero di riga i appartenga o meno alla regione r. Abbiamo già visto come fare a vedere se l'indirizzo di un byte appartiene o no ad una regione: è sufficiente guardare i $64-k$ bit più significativi dell'indirizzo di byte. In questo caso non abbiamo tutto l'indirizzo di byte, ma solo il numero di riga. Questo non è un problema: ci mancano solo 3 bit dell'indirizzo di byte, e questi sicuramente rientrano nei k che già dovevamo ignorare. Un altro modo per visualizzare l'operazione è di riportare tutto alle righe: una regione di 2^k byte è anche una regione di 2^{k-3} righe. I numeri di regione restano gli stessi che per i byte e per sapere se una riga appartiene o no alla regione scelta è sufficiente ignorare $k - 3$ bit meno significativi del numero di riga e confrontare gli altri con il numero di regione r. I $k - 3$ bit meno significativi, invece, sono l'offset *della riga* all'interno della regione. Questi possono essere usati per selezionare il byte corretto in ciascun chip di RAM.

Il circuito risultante è quello di Figura 3. La maschera M_r serve ad abilitare o disabilitare complessivamente tutta la scheda di memoria.

La parte interna al rettangolo tratteggiato in Fig. 3 rappresenta la scheda di memoria vera e propria, mentre la parte esterna rappresenta i circuiti del bus a cui la scheda è collegata. Il bus può proseguire a destra e sinistra e avere altre maschere che riconoscono valori di r diversi. Tipicamente avremo una "scheda madre" che prevede un certo numero di slot in cui si possono inserire le schede di memoria. Ogni slot avrà una maschera diversa. Le schede di memoria possono invece essere tutte uguali tra loro ed essere montate in un qualunque slot; più schede possono essere montate contemporaneamente sul bus, ciascuna ovviamente inserita in uno slot diverso.

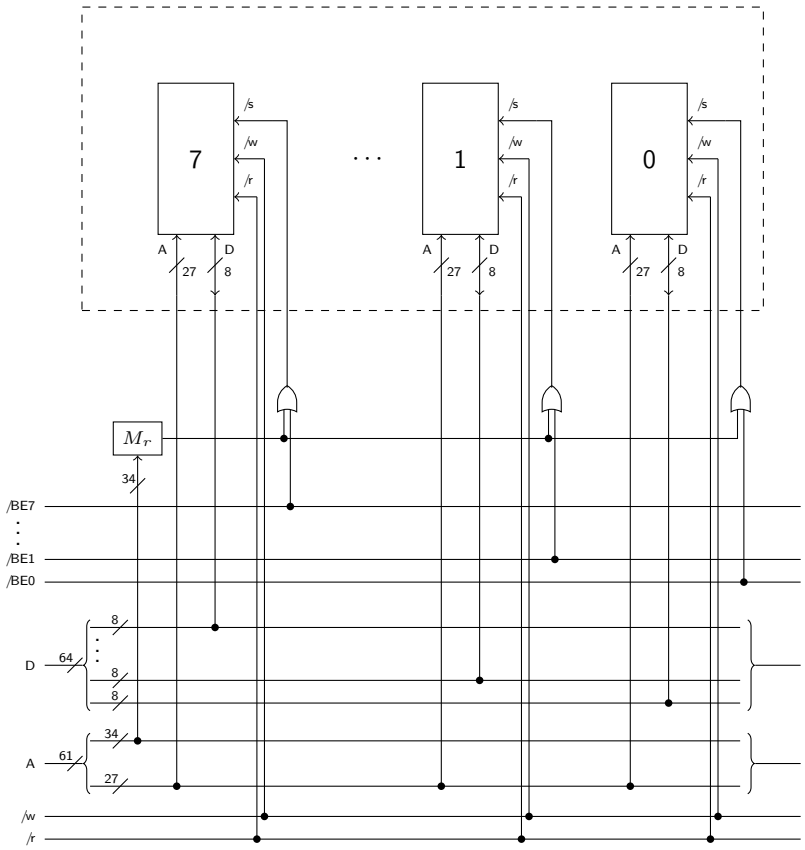

Figura 3: Maschera per il montaggio della memoria ad un particolare indirizzo.