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;
- 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*. Diamo un po' di definizioni:

- si dice che una entità contenuta in memoria è *allineata a n* (sottindendendo byte) se parte da un indirizzo che è multiplo di n;
- si dice che è *allineata a x*, dove x è una qualche entità, se è allineata alla dimensione (in byte) di x;
- infine, si dice che è *allineata naturalmente* se è allineata a sé stessa.

Quindi una parola quadrupla è allineata naturalmente se parte da un indirizzo che è multiplo della sua dimensione in byte, che è 8. 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 siano interamente contenute in una parola quadrupla allineata naturalmente.

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

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

Figura 1: Organizzazione della memoria su righe di 64 bit. "indirizzo LSB" (dove LSB sta per *Least Significant Byte*) è l'indirizzo del byte di indirizzo più basso nella riga. Gli indirizzi degli altri byte della riga si ottengono sommando a Ind. LSB il valore scritto in cima alla corrispondente colonna.

Conviene dunque rappresentare la memoria non come una sequenza di byte, ma come una sequenza di righe di 8 byte ciascuna (il numero massimo di byte che può essere trasferito in una singola operazione), come in Fig. 1. 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.

Per specificare completamente una richiesta di operazione di lettura dobbiamo presentare alla memoria l'indirizzo del primo byte da leggere 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 tramite dei fili di *numero di riga* e 8 fili di *byte enable*, come segue. In Fig. 1 abbiamo scritto sulla destra l'indirizzo del byte più a destra, mentre sulla sinistra abbiamo assegnato ad ogni riga un numero di riga. Il numero di riga non è altro che l'indirizzo di destra diviso per 8, operazione che in base 2 corrisponde a eliminare i 3 bit meno significativi. La memoria non riceve tutte le linee di indirizzo A63–A0, ma solo quelle che individuano il numero di riga, vale a dire A63–A3 [2]. Una volta selezionata in questo modo una riga, i singoli byte al suo interno sono selezionati ciascuno da una specifica linea di byte enable: per questo scopo avremo le linee /BE7–/BE0.

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 $A63 = A62 = \cdots = A10 = 0$, $A9 = 1$, $A8 = A7 = \cdots = A3 = 0$, in quanto 512 è 1000000000 in binario, e $/BE7 = /BE6 = \cdots = /BE0 = 0$ (tutti i byte abilitati);
- Lettura di una parola doppia all'indirizzo 512: A63–A3 come prima, ma $/BE7 = /BE6 = /BE5 = /BE4 = 1$ (byte 7–4 disabiliati) e $/BE3 = /BE2 = /BE1 = /BE0 = 0$ (byte 3–0 abilitati);

[2] A52–A3 per AMD64.

- Lettura di una parola doppia all'indirizzo 516: A63–A3 ancora come prima, in quanto siamo sempre all'interno della stessa riga. Avremo /BE7 = /BE6 = /BE5 = /BE4 = 0 (byte 7–4 abilitati) e /BE3 = /BE2 = /BE1 = /BE0 = 1 (byte 3–0 disabilitati).

Si noti che questo tipo di codifica permetterebbe di specificare molto di più di quanto abbiamo bisogno. Per esempio, con /BE7 = /BE5 = /BE3 = /BE1 = 1 e /BE6 = /BE4 = /BE2 = /BE0 = 0 potremmo leggere solo i byte di indirizzo 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 utilizzando 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.

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 (per esempio dal processore) continuano ad essere 2^{61}. Consideriamo una memoria composta da 2^a righe, per esempio $a = 30$ per una memoria di 8 GiB (in quanto avrà 8 dispositivi di 1 GiB ciascuno). Quello che possiamo fare è di scegliere un insieme contiguo di 2^a righe tra le 2^{61} disponibili e fare in modo che la nostra memoria risponda alle richieste che rientrano nel sottoinsieme scelto, e ignori 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). Più in generale, assumiamo che i numeri di riga disponibili siano 2^b, con $b \geq a$. È comodo che il sottoinsieme di indirizzi scelto parta da un indirizzo multiplo di 2^a. In questo caso il numero di riga su b bit viene a suddividersi naturalmente in due parti: la parte più significativa di $b-a$ bit e la parte meno significativa di a bit. Si veda la Fig. 3. Tutti i numeri di riga appartenenti al sottoinsieme scelto saranno uguali nei $b - a$ bit più significativi. Chiamiamo x il numero binario espresso da tali

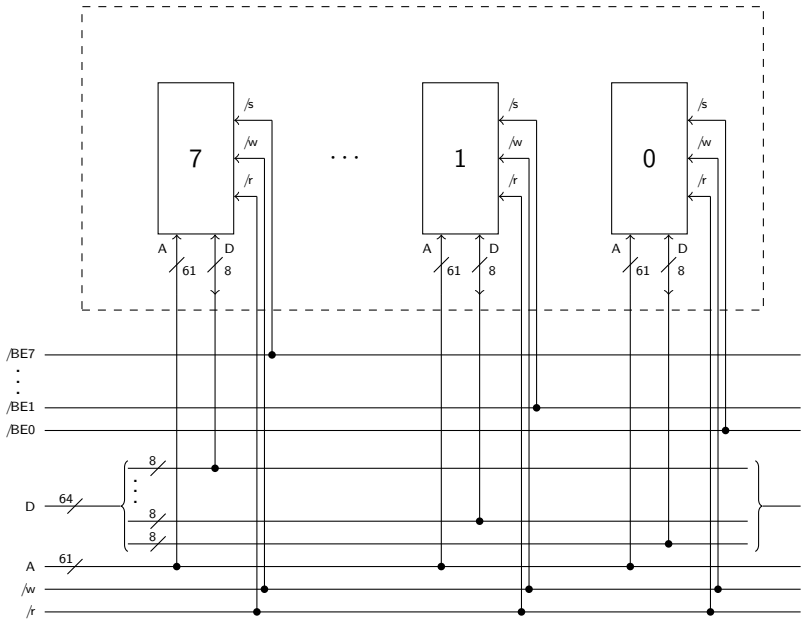

Figura 2: Realizzazione di una memoria organizzata a linee di 64 bit (parole quadruple), ma accessibile anche a byte, parole e doppie parole. Le parti dentro e fuori del rettangolo tratteggiato possono essere realizzate separatemente e poi collegate. La parte interna è una scheda di memoria, la parte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

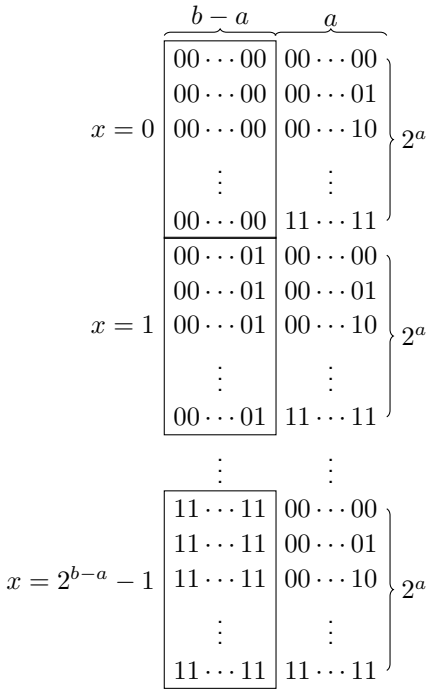

Figura 3: Scomposizione di un numero di riga di b bit in a bit meno significativi e $b - a$ più significativi.

bit. Se scegliamo il sottoisieme che parte dal numero di riga 0 avremo $x = 0$, se scegliamo il sottoinsieme immediatamente successivo avremo $x = 1$, e così via fino a $x = 2^{b-a} - 1$. I $b - a$ bit più significativi di tutti i numeri di riga che *non* appartengono al sottoinsieme scelto formerammo invece un numero diverso da x. Un'altra cosa da notare è che se guardiamo solo gli a bit meno significativi, ignorando i $b - a$ più significativi, tutti i sottonsiemi si comportano allo stesso modo, coprendo tutti i numeri di riga da 0 a $2^a - 1$. Questi sono proprio i numeri di riga che la nostra memoria di 2^a byte è in grado di decodificare.

Quindi la nostra memoria può essere collocata in uno qualunque dei sottonsiemi; scegliamone uno e chiamiamo $\bar{x}$ il numero espresso dai $b - a$ bit più significativi di tutti i suoi indirizzi. Data una operazione di lettura o scrittura ad un certo indirizzo i, la nostra memoria deve sapere se ignorarla o no in base al fatto che il numero di riga i appartenga o meno al sottoinsieme scelto. Per saperlo basta prendere il numero binario espresso dai $b - a$ bit più sigificativi di i, sia y, e confrontarlo con $\bar{x}$: se $y = \bar{x}$ allora i appartiene al sottoinsime, altrimenti no. Questa operazione può essere svolta tramite una "maschera" $M_{\bar{x}}$ che prende in ingresso i $b - a$ bit più significativi di i e restituisce un segnale che vale 0 se i bit sono "riconosciuti" (il numero da loro espresso è uguale a $\bar{x}$) e 1 altrimenti. A questo punto il circuito diventa come in Fig. 4, dove $a = 30$ e $b = 61$. Ogni dispositivo deve essere abilitato (/s pari a 0) se e solo se il corrispondente byte enable è abilitato e la maschera ha riconosciuto il numero di riga.

La parte interna al rettangolo tratteggiato in Fig. 4 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 x 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.

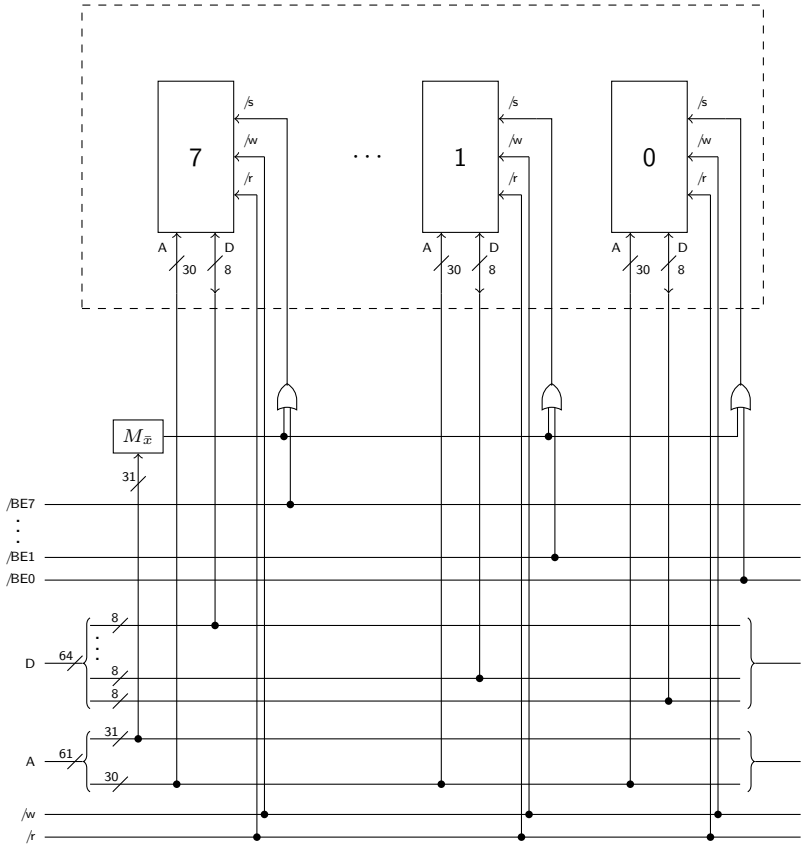

Figura 4: Maschera per il montaggio della memoria ad un particolare indirizzo.