Calcolatori Elettronici: indirizzi e oggetti

G. Lettieri

3 Marzo 2019

Gli indirizzi sono relativi ad un bus: tutti i componenti collegati al bus, in grado di rispondere a richieste di lettura o scrittura, devono avere degli indirizzi assegnati in modo univoco. I possibili indirizzi vanno praticamente sempre da 0 fino ad un massimo della forma $2^n - 1$ per qualche n che dipende dal bus. Questo perché gli indirizzi viaggiano su un numero prefissato di piedini, fili o tracce, ciascuna delle quali può assumere solo i valori 0 o 1. Se queste tracce sono n, il numero totale di indirizzi possibili è dunque 2^n.

Gli indirizzi esistono sempre tutti, nel senso che è sempre possibile richiedere una lettura o una scrittura a qualunque indirizzo, anche se l'indirizzo non è assegnato a nessun componente (il risultato di una tale richiesta dipende dal bus; come abbiamo detto, assumiamo per ora che le scritture non abbiano effetto e le letture restituiscano un valore casuale).

Per questi motivi, il modo più naturale di rappresentare gli indirizzi di un bus è come la sequenza di tutti i numeri binari, senza segno, su n bit. Conviene acquisire familiarità con le rappresentazioni dei numeri in base 16 e 8, in quanto queste basi permettono di rappresentare gli indirizzi in forma molto più compatta e, allo stesso tempo, facilmente convertibile da e verso la base 2. Alcuni numeri molto frequenti devono subito richiamare alla mente la loro rappresentazione nelle varie basi: in base 2, 2^a è un 1 seguito da a zeri mentre $2^a - 1$ è composto da a 1. Se a è un multiplo di 4, allora 2^a in base 16 è 1 seguito da $a/4$ zeri, mentre $2^a - 1$ è rappresentato come $a/4$ cifre F (questo perché ogni cifra esadecimale corrisponde a 4 cifre binarie). Similmente per la base 8: se a è multiplo di 3, allora 2^a è 1 seguito a $a/3$ zeri e $2^a - 1$ è composto da $a/3$ cifre 7. In generale, conviene usare queste basi ogni volta che dobbiamo ragionare sulla struttura binaria di un qualche valore. È bene familiarizzarsi con i seguenti casi notevoli:

- un byte corrisponde a 2 cifre esadecimali;
- 512 corrisponde a 1000 in base 8;
- 4 Ki corrisponde a 1000 in base 16 e 1.0000 in base 8;
- 1 Mi corrisponde a 10.0000 in base 16;
- 4 Gi corrisponde a 1.0000.0000 in base 16.

Le operazioni sugli indirizzi (come aggiungere una costante a un indirizzo, o sottrarre due indirizzi) vanno sempre pensate come operazioni modulo 2^n. Questo comporta che l'indirizzo che precede l'indirizzo 0 è l'indirizzo $2^n - 1$, e l'indirizzo successivo a $2^n - 1$ è l'indirizzo zero.

1 Scostamenti (*offset*)

Dati due indirizzi x e y su n bit, possiamo chiederci quanto y *si discosta* da x calcolando il valore $y - x$ modulo 2^n, detto *offset* di y rispetto a x.

In ogni caso, l'offset conta il numero di indirizzi che è necessario "saltare", partendo da x, per raggiungere y. In particolare, l'offset di x rispetto a se stesso è zero. Gli offset possono essere anche negativi: il loro valore assoluto rappresenta comunque il numero di indirizzi da saltare partendo da x per raggiungere y, ma andando nella direzione degli indirizzi descrescenti. Per esempio, l'offset -1 rappresenta l'indirizzo che precede x (nel caso $x = 0$ si ricordi che l'indirizzo precedente è $2^n - 1$).

Se rappresentiamo gli offset come numeri in complemento a 2 su n bit, diventa indifferente considerarli con o senza segno. Facciamo un esempio con $n = 4$. Gli indirizzi vanno dunque da 0 a 15. L'offset -1 si rappresenta come 1111 in complemento a 2 su 4 bit. Prendiamo $x = 3$. Se interpretiamo 1111 come numeo con segno, l'offset è -1 e saltando un indirizzo all'indietro arrivamo a $y = 2$. Se ora interpretiamo l'offset 1111 come il numero senza segno 15, dobbiamo saltare 15 indirizzi in avanti partendo da $x = 3$. Dopo averne saltati 12 arrivamo all'indirizzo 15, saltandone un altro ripartiamo dall'indirizzo 0 e con gli ultimi due salti arriviamo a $y = 2$, come prima. Si noti, però, che l'equivalenza vale solo se l'offset è rappresentato su un mero di bit maggiore o uguale n. Se è rappresentato su meno bit, è necessario sapere se va interpretato come numero con o senza segno.

2 Intervalli (*range*)

Un *intervallo* è una sequenza di indirizzi. Conviene quasi sempre rappresentarlo specificando il primo indirizzo che fa parte dell'intervallo, sia x, e il primo, successivo a x, che *non* ne fa parte, sia y. In altre parole, come un intervallo di numeri aperto a destra $\{ n \mid x \leq n < y \}$, che si rappresenta più concisamente come $[x, y)$. Si noti che, in base a questa definizione, un intervallo $[x, y)$ in cui $y \leq x$ è vuoto. Ci limitiamo a considerare intervalli $[x, y)$ in cui $y \geq x$[1]

Uno dei vantaggi di questa scelta è che l'offset tra y e x (in altre parole, $y-x$) rappresenta sempre il numero di indirizzi che fanno parte dell'intervallo. Questo vale anche per l'intervallo $[x, x)$, che è vuoto. Se avessimo scelto di includere anche l'estremo di destra avremmo dovuto invece sommare 1 alla differenza dei due estremi.

[1] Per seplicità, dunque, evitiamo di considerare intervalli che attraversano l'ultimo indirizzo rappresentabile e ripartono da zero. Questi avrebbero $y < x$.

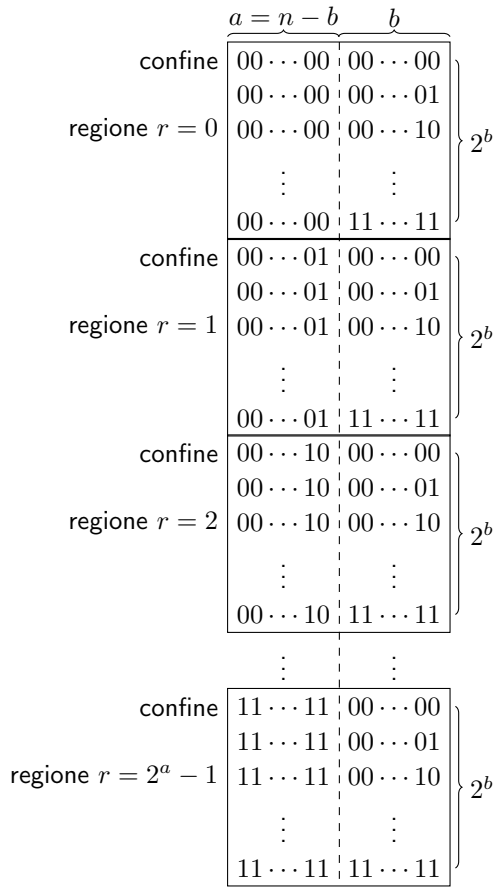

Figura 1: Scomposizione degli indirizzi in regioni naturali di 2^b.

L'indirizzo x è detto *base* dell'intervallo. Se di un intervallo conosciamo la base x e la lunghezza l, l'intervallo è dato da $[x, x + l)$. Si noti ancora l'assenza di fastidiosi -1.

D'altra parte, nella notazione aperta a destra bisogna ricordarsi che l'ultimo indirizzo che *fa parte* dell'intervallo $[x, y)$ è $y - 1$ (l'indirizzo che precede y, che è il primo che non ne fa parte).

3 Confini (*boundaries*) e regioni naturali

Dato un qualunque $b \leq n$, tutti gli indirizzi multipli di 2^b sono detti *confini* di 2^b. I confini di 2^b terminano con almeno b zeri nella loro rappresentazione in base 2.

Si noti che l'indirizzo zero è un confine per qualunque b. Tutti i confini si trovano quindi partendo da 0 e procedendo ad offset successivi di 2^b. Questi confini delimitano degli intervalli di indirizzi che chiamiamo *regioni naturali* di 2^b. Ometteremo l'agettivo "naturali" quando non si crea ambiguità. Useremo il termine regione (naturale) in modo generico, ma in molti casi introdurremo dei nomi particolari per regioni di particolare interesse (per esempio, *cacheline*, *pagina*, ...). Le regioni sono in totale 2^{n-b}, o 2^a se poniamo $a = n-b$ (Figura 1). Incidentalmente, notiamo che se un indirizzo x è un confine per 2^b è anche un confine per qualunque $b' < b$.

Possiamo assegnare ad ogni regione un numero progressivo r compreso tra 0 e $2^a - 1$. Il numero r, detto *numero di regione*, serve ad identificare univocamente ogni regione. Si noti che tutti (e soli) gli indirizzi che fanno parte della stessa regione contengono negli a bit più significativi proprio il numero della regione a cui apparengono. Invece, i b bit meno significati contengono l'offset dell'indirizzo rispetto alla base della regione (il confine), detto *offset all'interno della regione*. Ogni indirizzo è dunque identificato dal numero di regione e dal suo offset (all'interno della regione). Ovviamente, lo stesso indirizzo può essere identificato in modi diversi in base alla dimensione della regione scelta.

Dato un numero b e un indirizzo x su n bit è dunque immediato, in hardware, trovare il suo numero di regione (di 2^b) e il suo offset: gli $a = n - b$ bit più significativi x danno il numero di regione, e i b bit meno significativi danno l'offset. Supponiamo ora di voler fare la stessa cosa in software, supponendo di avere una variable **`unsigned long`** `x` che contiene un undirizzo, e di conoscere b. Prendiamo come n la dimensione in bit di un **`unsigned long`**, che nel nostro caso è 64. Per ottenere il numero di regione possiamo scrivere semplicemente

```
r = x >> b; // numero di regione
```

Per ottenere l'offset usiamo l'AND bit a bit con una maschera che ha b cifre meno significative ad 1 e tutte le altre a zero:

```
o = x & ((1UL << b) - 1); // offset
```

I caratteri `UL` che seguono la costante 1 servono a specificarne il tipo come Unsigned Long. Si ricordi che per default le costanti numeriche del C++ hanno tipo **`int`**.

Vediamo anche come si ottiene *l'indirizzo* della regione a cui x appartiene. Per indirizzo della regione intendiamo il confine da cui parte. Possiamo ottenere questo indirizzo con una maschera che ha a bit significiativi a 1 e gli altri a 0. Questa maschera non è altro che il NOT della maschera che abbiamo usato prima:

```
c = x & ~((1UL << b) - 1); // confine
```

Dato un intervallo $[x, y)$, vogliamo spesso sapere quali sono la prima e l'ultima regione toccate dall'intervallo. Per farlo è sufficiente operare come prima, usando gli indirizzi x (prima toccata) e $y - 1$ (ultima toccata).

4 Oggetti e allineamenti

Un *oggetto* è una sequenza di un certo numero di byte, sia l. Possiamo assegnare indirizzi a tutti i byte di un oggetto scegliendo l'indirizzo del suo primo byte e assegnando gli altri indirizzi in sequenza. Se x è l'indirizzo del primo byte dell'oggetto, questo viene ad occupare l'intervallo $[x, x + l)$. L'indirizzo x è quasi unanimente considerato l'indirizzo dell'oggetto stesso (e non solo del suo primo byte) e anche noi faremo così.

Diamo un po' di definizioni:

- si dice che un oggetto o è *allineato a* 2^b (sottindendendo byte) se il suo indirizzo è un confine di 2^b;
- se la dimensione di o' è una potenza di 2, si dice che o è *allineato a* o' se è allineato alla dimensione di o';
- infine, se la dimensione di o è una potenza di 2, si dice che o è *allineato naturalmente* se è allineato a se stesso.

Un oggetto allineato naturalmente occupa interamente una regione della sua dimensione.

Si noti che l'allineamento non è una proprietà dell'oggetto, ma dell'indirizzo che gli è stato assegnato. Lo stesso oggetto può essere allineato a dimensioni diverse in base a dove si trova.