Appunti di Architettura

Architettura degli Elaboratori (prof. Fersini) - CdL Informatica Unimib - 23/24

Federico Zotti

08 Mar 2024

Indice

1 Sistemi Numerici

1.1 Introduzione

I calcolatori utilizzano, a differenza di noi, il **sistema binario**. Questo perché la corrente elettrica può rappresentare solo due stati: acceso (*1*) e spento (*0*).

Sono stati definiti degli **standard di codifica**: regole che vengono utilizzate nella rappresentazione dei dati in formato binario.

Con il termine **bit** definiamo l'**unità di misura dell'informazione**. Combinando tra loro più bit si ottengono strutture più complesse. In particolare:

- Nybble (o nibble): 4 bit
- Byte: 8 bit
- Halfword: 16 bit
- Word: 32 bit
- Doubleword: 64 bit

Dati k bit, il numero di configurazioni ottenibili è pari a 2^k.

Una **rappresentazione** è un modo per descrivere un'entità. Bisogna distinguere l'entità (o valore) e la sua rappresentazione.

1.2 Vari sistemi numerici

Il **sistema numerico decimale**:

- Usa 10 cifre
- È un **sistema posizionale**: ogni cifra assume un valore diverso a seconda della posizione che occupa all'interno del numero

Il **sistema romano** invece non è posizionale (il valore della cifra non dipende dalla sua posizione).

Nei sistemi numerici posizionali un valore numerico N è caratterizzato dalla seguente rappresentazione:

$$N = d_{n-1}\, d_{n-2} \dots d_1\, d_0\,,\, d_{-1} \dots d_{-m}$$

$$N = d_{n-1} \cdot r^{n-1} + \dots + d_0 \cdot r^0 + d_{-1} \cdot r^{-1} + \dots d_{-m} \cdot r^{r-m}$$

$$N = \sum_{i=-m}^{n-1} d_i \cdot r^i$$

Dove: - d è la singola cifra - r la radice o base del sistema - n numero di cifre della parte intera (sinistra della virgola) - m numero di cifre della parte frazionaria (destra della virgola) - N è la rappresentazione del numero

1.2.1 Sistema Binario

Un **byte** è una sequenza di 8 bit consecutivi. Il bit *più a sinistra* è chiamato **MSB** (most significant bit) ovvero il bit che rappresenta la cifra con il valore più grande. Il bit *più a destra* è chiamato **LSB** (least significant bit) ovvero il bit che rappresenta la cifra con il valore più piccolo.

2 Rappresentazione di numeri interi con segno

2.1 Operazioni aritmetiche

2.1.1 Somma

La somma di due sequenze di bit è la somma tra i bit di pari ordine:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$ con riporto 1 sul bit di ordine superiore
- $1 + 1 + 1 = 1$ con riporto 1 sul bit di ordine superiore

Dunque la somma è definita su 3 elementi:

- I due addendi
- Il riporto (carry)

2.1.2 Sottrazione

La sottrazione di due sequenze di bit è la sottrazione tra i bit di pari ordine:

- $0 - 0 = 0$
- $1 - 0 = 1$
- $1 - 1 = 0$
- $0 - 1 = 1$ con prestito 1 dal bit di ordine superiore

Dunque la sottrazione è definita su 3 elementi:

- Minuendo e sottraendo
- Il prestito (borrow)

2.2 Modulo e Segno

Supponiamo di avere a disposizione 1 Byte per rappresentare numeri sia positivi che negativi.

Con il metodo del **modulo e segno** utilizzeremo:

- I primi 7 bit per il valore assoluto del numero
- Il bit più a sinistra *(MSB)* per indicare il segno (1 se il numero è negativo, 0 se è positivo)

Con n bit totali si possono rappresentare i numeri interi nell'intervallo

$$\left[-(2^{n-1}-1), +(2^{n-1}-1)\right]_{10}$$

I problemi di questa rappresentazione sono che esistono due diverse rappresentazioni dello 0 e che un bit viene "speso" solo per il segno.

2.2.1 Somma

Confronto i bit di segno dei due numeri:

- Se i bit di segno sono uguali:

  - Il bit di segno risultante sarà il bit di segno dei due addendi
  - Eseguo la somma bit a bit (a meno di overflow)

- Se i bit di segno sono diversi:

  - Confronto i valori assoluti dei due addendi
  - Il bit si segno risultate sarà il bit di segno dell'addendo con valore assoluto
  - Eseguo la somma bit a bit

2.2.2 Sottrazione

Confronto i bit di segno dei due numeri:

- Se i bit di segno sono uguali:

  - Il bit di segno risultante sarà uguale al bit di segno dell'operando a modulo maggiore
  - Il risultato avrà modulo pari al modulo della differenza dei moduli degli operandi

- Se i bit di segno sono diversi

  - Il bit di segno risultante sarà uguale al bit di segno del minuendo
  - Il risultato avrà il modulo pari alla somma dei moduli dei due operandi

2.2.3 Overflow

Si può avere overflow solo quando:

- Si sommano due operandi con segno concorde
- Si sottraggono due operandi con segno discorde

2.3 Complemento a 1

Questo metodo si basa sull'**operazione di complemento**. Con complemento si intende l'operazione che associa ad un bit il suo opposto.

Il metodo del **complemento a 1** è molto semplice:

- Se il numero da codificare è **positivo**, lo si converte in binario con il metodo tradizionale
- Se il numero è **negativo**, basta convertire in binario il suo modulo e quindi eseguire l'operazione di complemento sulla codifica binaria effettuata

Anche in questo caso sussiste il problema delle due diverse rappresentazioni dello 0.

2.4 Complemento a 2

Il complemento a 2 è un altro metodo di codifica usato per rappresentare i numeri interi sia positivi che negativi. È basato sul complemento a 1.

A differenza del complemento a 1 i numeri negativi dopo aver complementato il numero vengono incrementati di 1.

Dati n bit si possono rappresentare i numeri interi nell'intervallo

$$\left[-(2^{n-1}), +(2^{n-1}-1)\right]_{10}$$

2.4.1 Conversione da CA2 a decimale

Se il numero è positivo (MSB = 0), si converte in base decimale usando il numero binario puro. Se il numero è negativo (MSB = 1), si applica l'operazione di CA2 a questo valore ottenendo la rappresentazione del corrispondente positivo, si converte il risultato come numero in binario puro e si aggiunge il segno meno.

2.4.2 Somma

1. Si esegue la somma su tutti i bit degli addendi, segno compreso
2. Un eventuale riporto oltre il bit di segno (MSB) viene scartato
3. Nel caso gli operandi siano di segno concorde occorre verificare la presenza o meno di overflow (si presenta solo se $(+A)+(+B)=-C$ o $(-A)+(-B)=+C$)

2.4.3 Sottrazione

La sottrazione tra due numeri in CA2 viene trasformata in somma applicando la regola

$$A - B = A + (-B)$$

2.5 Shift

Nel caso lo shift sia con un numero MS il segno non viene considerato e viene mantenuto.

Nel caso lo shift sinistro sia con un numero CA2 il segno non viene considerato (si sposta come tutti gli altri bit) e se il nuovo MSB è diverso dal precedente c'è un overflow. Nello shift destro il segno viene replicato.

3 Rappresentazione numeri reali e altre informazioni

I numeri reali possono essere rappresentati sia in virgola fissa *(fixed point)* che in virgola mobile *(floating point)*.

3.1 Numeri in virgola fissa

Un sistema di numerazione in **virgola fissa** è quello in cui si riserva un numero fisso di bit per la parte intera e la parte frazionaria. La posizione della virgola è **implicita** e uguale per tutti i numeri.

3.1.1 Unsigned fixed point

Per i numeri **unsigned** fixed point, dati n bit a disposizione

- $i < n$ bit per rappresentare la **parte intera** del numero
- $d = n - i$ bit per rappresentare la **parte decimale** del numero

Con questo metodo l'intervallo di numeri interi rappresentabili è

$$[0, 2^i - 1]$$

e l'intervallo rappresentabile dalla parte decimale è

$$[0, 2^d - 1]$$

3.1.2 Signed fixed point

Per i numeri **signed** fixed point, dati n bit a disposizione

- Un bit per il segno del numero da rappresentare
- $i < (n - 1)$ bit per rappresentare la **parte intera** del numero
- $d = n - (i + 1)$ bit per rappresentare la **parte decimale** del numero

Con questo metodo l'intervallo di numeri interi rappresentabili è

$$[-2^{i-1} - 1, \quad 2^{i-1} - 1]$$

e l'intervallo rappresentabile dalla parte decimale è

$$[0, 2^d - 1]$$

3.2 Numeri in virgola mobile

Nella notazione in **virgola mobile** un numero N è esprimibile come

$$N = \pm M \cdot B^{\pm E}$$

Vengono usati:

- Un bit per il segno
- n bit per la mantissa

- m bit per l'esponente

Dunque la vera rappresentazione in binario sarà

$$N = (-1)^S \cdot M \cdot B^E$$

Il numero rappresentato deve essere **normalizzato**, ovvero viene trasformato utilizzando solo una cifra intera:

$$1101.10011 \rightarrow 1.110110011$$

Dunque essendo il primo bit (la parte intera) sempre 1, non viene memorizzato e viene definito come **bit nascosto**.

Per rappresentare -53.5 in floating point 32 bit:

$$-53.5 = (-110101.1)_2 = (-1)^{(1)_2} \cdot (1.101011)_2 \cdot 2^{(101)_2}$$

```
|   1    |  00000101   |  10101100000..00   |
| Segno  |  Esponente  |      Mantissa      |
```

Lo standard che definisce la rappresentazione dei numeri in virgola mobile è lo **IEEE 754** (1985). Specifica il formato, le operazioni e le conversioni tra i diversi formati floating point e quelle tra i diversi sistemi di numerazioni.

Secondo questo standard l'esponente ha 8 bit (rappresentato in eccesso 127, polarizzato). I valori estremi -127 e 128 sono riservati. Dunque il numero più grande che può essere rappresentato (dall'esponente) è `111...111` e il più piccolo `000...000`. Per confrontare due esponenti basta considerarli interi senza segno.

Quindi in IEEE 754 un numero N in virgola mobile viene rappresentato come

$$N = (-1)^S \cdot (1 + 0.M) \cdot 2^{E-127}$$

3.2.1 Errore assoluto e Errore relativo

Rappresentando un numero reale n in virgola mobile si commette un errore di approssimazione. Questo perché viene rappresentato un numero razionale n' con un numero limitato di cifre significative.

L'errore assoluto è definito come

$$e_A = n - n'$$

e l'errore relativo come

$$e_R = \frac{e_A}{n} = \frac{n - n'}{n}$$

L'ordine di grandezza dell'errore assoluto dipende dal numero di cifre significative e dall'ordine di grandezza del numero. L'ordine di grandezza dell'errore relativo dipende solo dal numero di cifre significative.

3.3 Rappresentazione di caratteri

Possiamo associare ad ogni carattere un numero. Dunque possono essere rappresentati con diverse codifiche:

- ASCII standard: un carattere è rappresentato con 7 bit (128 simboli totali)
- ASCII esteso: un carattere è rappresentato con 8 bit (256 simboli totali)
- Unicode: un carattere è rappresentato con un numero maggiore di bit (da 8 a 32 bit per carattere)

4 Logica combinatoria

> Prendere dalle slide #todo-uni. Le porte e i livelli logici sono abbastanza banali.

> Tutto prima del decoder.

