La codifica binaria della informazione

Vitoantonio Bevilacqua

bevilacqua@poliba.it

Sommario. Il presente paragrafo si riferisce alle prime due lezioni del corso di Fondamenti di Informatica e Laboratorio di Informatica.

Parole chiave: Codifica Binaria, Memoria RAM, Complemento a 2, IEEE 754 a singola e doppia precisione.

1 Introduzione

Il fine principale degli argomenti trattati in queste iniziali lezioni consiste nell'illustrare in quale maniera l'informazione, per adesso soltanto numerica, viene memorizzata nella memoria RAM (Random Access Memory) di un sistema di calcolo rispettando standard condivisi, per poi essere elaborata attraverso un linguaggio di programmazione. In particolare si tratteranno: la codifica binaria, la codifica esadecimale, le conversioni di base, il concetto di errore di una codifica, la codifica di numeri interi con e senza segno in CA2 (complemento a 2), la codifica di numeri reali in singola e doppia precisione secondo lo standard IEEE 754, gli effetti sulla dichiarazione delle variabili signed e unsigned di tipo char ed int, e delle variabili float e double in linguaggio C.

2 La memoria di lavoro

La memoria RAM (Random Access Memory) è la memoria di lavoro (elaborazione) di un sistema di calcolo; per semplicità essa può essere rappresentata come una tabella organizzata in righe, ciascuna delle quali (per ora chiamata word) viene suddivisa in 4 colonne o blocchi, da ora in poi chiamati byte, costituiti da una sequenza di 8 bit (binary digit) ovvero 8 cifre che possono assumere soltanto valori 0 o 1.
In generale, ogni programma di elaborazione si occupa della gestione delle informazioni per unità elementari corrispondenti alla dimensione dei byte, senza necessariamente scendere al livello di dettaglio dei singoli bit, per questo motivo si dice che la unità minima indirizzabile (ovvero dotata di un indirizzo in memoria corrispondente alla posizione in memoria RAM) è il singolo byte. Il modo più diffuso di numerare i byte, iniziando sempre dal numero progressi 0, è da destra verso a sinistra (“little Endian”), laddove il verso dall’alto verso il basso è ovviamente relativo a come più avanti si dirà essere partizionata la intera memoria RAM.

3 Le regole per le trasformazioni di base

Dato un numero, le basi del sistema di numerazione che si usano per leggere tale numero sono la base 2 (binaria), 8 (ottale), 16 (esadecimale); assegnata una base del sistema di numerazione, le cifre sono date da 0 fino a numero base – 1. Dopo la cifra 9, si aggiungono A,B,C,D,E,F per formare la base di un sistema esadecimale.

Assegnato un numero codificato in base 2, si decodifica facilmente in base 10; esempio della codifica chiamata "in binario puro":
$(1010.101)_2=(1 \times 2^3+0 \times 2^2+1 \times 2^1+0 \times 2^0+1 \times 2^{-1}+0 \times 2^{-2}+1 \times 2^{-3})_{10}=(10.625)_{10}$

Codifica binaria di un numero: per l'operazione inversa, si utilizzano due algoritmi rispettivamente per la parte intera e decimale del numero:

3.1 Algoritmo di Horner - divisioni successive

Si prende il numero intero, si mette in colonna a sinistra, lo si divide per 2, riportando quoziente sotto il dividendo mentre resto accanto al dividendo, e così via fino a raggiungere il quoziente 0. La parte intera del numero in binario è la sequenza dei resti, presa dal basso verso l'alto.

10	0
5	1
2	0
1	1
0	

3.2 Algoritmo di Horner - moltiplicazioni successive

Si prende il numero decimale, si mette in colonna a sinistra, lo si moltiplica per 2, riportando

1. se il prodotto è ≥1, accanto al primo fattore l'1 e sotto il complemento del prodotto
2. se il prodotto è <1, accanto al primo fattore lo 0 e sotto il prodotto

e così via fino a raggiungere il prodotto 0. La sequenza da prendere come parte decimale del numero in binario è la sequenza di destra presa dall'alto verso il basso.

.625	1
.25	0
.5	1
0	

Mentre nell'algoritmo delle divisioni lo 0 si raggiunge sempre, nell'algoritmo delle moltiplicazioni non sempre è possibile raggiungerlo e ciò dipende dal concetto di periodo della base; il periodo si ripete dal momento in cui si trova un numero già trovato in precedenza

.4	0
.8	1
.6	1
.2	0
.4	…
…	

Esempio: codificando in binario e ricodificando in decimale il numero 127.4

$$(127.4)_{10}=(1111111.0110)_2=(127.375)_{10}$$

ciò perché nel momento in cui si accetta un periodo, si commette un "errore di codifica".

Si definisce:

errore assoluto e_a = scarto tra numero iniziale da codificare e numero ottenuto dopo la ricodifica nella stessa base di partenza; es: e_a=127.4-127.375=0.025

errore relativo $e_r = e_a$/(numero iniziale rispetto al quale si è commesso l'errore di codifica); es: e_r=0.025/127.4

errore relativo percentuale $e_{r\%}=e_r \cdot 100$

Concetto di "range" (=intervallo di rappresentazione di un numero): preso un byte, supponendo che, avendo a disposizione n bit, tutti gli n codificano numeri interi in base binaria, il range di rappresentazione varia tra $[0,2^n-1]$; la suddetta formula di range deriva dal numero di combinazioni diverse che si possono ottenere, 2^n=256, tenendo conto che si inizia la numerazione da 0. Dunque essendo 8 i bit, range : [0,255].

Codifica esadecimale di un numero: è necessaria per minimizzare il codice; per eseguirla si possono utilizzare due metodi:

a. gli algoritmi da utilizzare sono sempre gli algoritmi di Horner, con le opportune modifiche (in colonna si divide e si moltiplica per 16); infatti i due algoritmi sono generici nel senso che non dipendono dalla base in cui è rappresentato un numero.

b. si codifica in binario il numero, quindi essendo $16=2^4$, si raccolgono le cifre a 4 a 4, a partire dalla virgola, e si decodifica da base binaria a base decimale ciascuno dei blocchi

Esempio: codifica in esadecimale di 65.25:

$(65.25)_{10}=(1000001.01)_2$;

0100	0001	.	0100
⇩	⇩		⇩
4	1	.	4

dunque $(65.25)_{10}=(41.4)_{16}$

3.3 Rappresentazione con modulo e segno ("signed magnitude" o M&S)

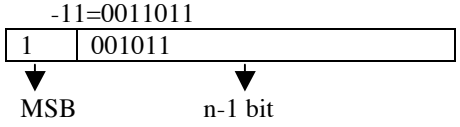
Per rappresentare un numero con segno, si usa, per convenzione:

- un bit per il segno: 0 per +, 1 per –
- n-1 bit per il modulo

Il bit che codifica il segno è definito MSB (Most Significant Bit, il primo bit).

Esempio: n=8 bit , range [-15,+15]

-11=0011011



In questo caso però le combinazioni si riducono a 255 causa la doppia rappresentazione dello 0: 10000000=00000000=0; in questo caso il range varia in $[-(2^{n-1}-1),2^{n-1}-1]$, [-127,127].

3.4 L'operazione di complemento a uno. (CA1)

Il complemento a uno è un'operazione binaria preliminare alla rappresentazione in CA2, quest'ultima utilizzata per rappresentare i numeri interi negativi e positivi. Tale operarazione, molto semplice per un sistema di calcolo, consiste nella "complementazione" di ciascun bit, ossia nell'inversione dei bit.

3.5 La codifica in complemento a due per i numeri con segno (CA2)

Si è già detto, che per i numeri interi con segno, il precedente metodo in modulo e segno si è dimostrato essere poco efficiente, per ovviare a ciò, si considera la :

Rappresentazione in C.A.2: il valore da utilizzare relativamente al MSB risulta essere il suo opposto (questo implica che se vale 0 porta a una decodifica di valore uguale alla rappresentazione in binario puro):

00000000=0
00000001=1
…
01111111=+127
10000000=$-1\cdot 2^7$=-128
10000001=-128+1=-127
…
11111111=-1

Le combinazioni sono 256 per la rappresentazione univoca dello 0: ovvero soltanto 00000000=0; in questo caso il range varia in $[-(2^{n-1}),2^{n-1}-1]$, [-128,127].

Per codificare un numero in complemento a due:

1. si sceglie il range più stretto che contiene il numero da codificare e di conseguenza il numero di bit necessari
2. si rappresenta il valore assoluto del numero in binario
3. si applica l'operazione di complemento a uno e si somma 1
4. la sequenza ottenuta è la codifica del numero iniziale in complemento a due

Esempio:
-65 codificato secondo CA2:

range: [-128;127]; 2^8 combinazioni; 8 bit necessari
v.a. 65 = 01000001

c.a1: 10111110

c.a2: 10111110+1=10111111

4 Lo standard IEEE 754

Standard IEEE 754 single precision: rappresentazione di un numero reale in 32 bit

I 32 bit vengono suddivisi in 3 gruppi:

±	esponente	mantissa
1	8	23

Il modulo del numero viene codificato in base binaria utilizzando gli algoritmi di divisione e moltiplicazione successive, evidenziando nella parte decimale, ove ci fosse, un periodo. La rappresentazione binaria che ne vien fuori è detta "fixed point" (a virgola fissa).

Si passa alla rappresentazione "floating point" (a virgola mobile) per determinare mantissa ed esponente. Si sposta la virgola fino alla sinistra dell'ultima cifra diversa da 0, cambiando anche l'esponente per riequilibrare l'ordine di grandezza; la parte dopo la virgola indica la mantissa matematica. Se si sposta indietro la virgola, alla destra dell'ultima cifra diversa da 0, riaggiustando l'esponente:

1. la parte di numero dopo la virgola indica la "mantissa normalizzata"; se la mantissa è composta da numero finito di cifre <23 (no periodo in algoritmo delle moltiplicazioni), ai restanti bit si assegna valore 0.
5. per codificare l'esponente, si usa la "rappresentazione per eccesso": si somma l'esponente al "bias" ($=2^{n-1}-1$ dove n è il numero di bit riservati all'esponente) quindi si codifica il numero utilizzando la rapp. dei numeri interi.

Esempio:
-118.625 codificato secondo IEEE754 a single precision:

parte intera: 118 = 1110110
parte decimale: 0.625 = .101
→ 1110110.101 fixed point

$0.1110110101 \cdot 2^7$
mantissa math: 1110110101

$1.110110101 \cdot 2^6$
mantissa norm: 110110101

esponente:
$6+2^7-1 = 6+127 = 5+128 =10000101$

rappresentazione single precision:

1	10000101	11011010100000000000000

L'ordine di precisione dello standard IEEE754 single è dato dal numero di cifre della mantissa. Per ottenere una rappresentazione più precisa esiste lo:

Standard IEEE 754 double precision: rappresentazione di un numero reale in 64 bit

I 64 bit vengono suddivisi in 3 gruppi:

±	esponente	mantissa
1	11	52

Il metodo di rappresentazione dell'IEEE754 double è analogo al single; l'unica cosa che varia è il bias: $2^{11-1}-1=1023$

Rappresentazione di ∞, infinitesimi e NaN nello standard IEEE754 single precision:
Delle 256 combinazioni dell'exp, nel caso in cui:

exp	mantissa	numero rappresentato
11111111	000000… (tutti 0)	$\pm\infty$
	010100… (non tutti 0)	NaN
00000000	000000… (tutti 0)	0
	010000… (non tutti 0)	infinitesimo con ordine di grandezza anche inferiore, in funzione del primo bit non nullo della mantissa, a 2^{-127}

Osservazioni:

- l'ordine di grandezza del numero dipende solo dall'esponente, essendoci l'1 davanti la mantissa nella rappresentazione binaria del numero; quindi
  - l'ordine di grandezza dell'infinito è $2^{255-bias}=2^{128}$
  - l'ordine di grandezza dello 0 o di un infinitesimo è 2^{-127}; tuttavia, se si impone al calcolatore di de normalizzare la mantissa, si ottiene uno 0 o comunque un infinitesimo di ordine ancora inferiore.
- l'ordine di grandezza dell'infinito nello standard IEEE754 single è minore dell'ordine di grandezza dell'infinito nello standard IEEE754 double

5 Verso il linguaggio C

Per specificare il tipo di variabile (tipo di rappresentazione usata per la codifica della variabile) esistono istruzioni dette "specificatori di tipo":

- "char a": carattere = variabile a memorizzata in 1 byte, codificata secondo C.A.2
- "int b": intero = variabile b memorizzata in 2 o 4 byte (4 in VisualC++ 6.0), codificata secondo C.A.2
- "float c": single = variabile c memorizzata in 4 byte, codificata secondo IEEE754 single p.
- "double d": double = variabile d memorizzata in 8 byte, codificata secondo IEEE754 double p.

Non indicando niente davanti agli specificatori di tipo, si sottintende "signed".
Per specificare che la variabile è senza segno, si scrive "unsigned" davanti agli specificatori di tipo e la codifica avverrà senza segno.
Per dimensionare una variabile, esistono gli specificatori "short" e "long" da scrivere davanti agli specificatori di tipo per rispettivamente dimezzare o raddoppiare il numero di byte necessari alla rappresentazione.
Per assegnare ad una variabile a il valore n si utilizza l'assegnazione " a = n ; " e, a seconda del tipo specificato di variabile, il calcolatore memorizzerà il valore n nel primo pacchetto libero di byte in modo che il primo byte sia multiplo di 4.
L'indirizzo di una variabile "a" è indicato con "&a" ed è quindi multiplo di 4.
N.B. la riga di codice a=a+1 ; non è un'uguaglianza, ma un'assegnazione! : il calcolatore all'indirizzo &a sovrascriverà al valore della variabile a, che quindi sarà perso, il valore incrementato di 1 .

Ringraziamenti. Il presente capitolo è stato scritto anche grazie al prezioso contributo dello studente Donato Mancuso con la successiva revisione del suo collega Pasquale Bonasia.

Riferimenti

1. Bevilacqua, V.: Dispense Teoria 1 e IEEE 754 In: http://www.vitoantoniobevilacqua.it
2. http://it.wikipedia.org/wiki/IEEE_754
3. http://it.wikipedia.org/wiki/Ordine_dei_byte

Appendice: Codice in linguaggio C (per ora solo di test)

```
#include <stdio.h>
#include <conio.h> /* non ANSI C */
/*#include <math.h>  per usare la funzione pow*/
void main()
{
            unsigned char byte[4]; unsigned char *c;
            char bit[9]; int i,j;float a;float *p=NULL;
            a=0;
            printf("Le dimensioni in byte di variabili \n");
            printf("char, int, double e float valgono \n");
printf("%d %d %d %d\n",sizeof(char),sizeof(int),sizeof(float),sizeof(double));
            do
            {
            printf("inserisci a = ");
            scanf("%f",&a);
/*a=(float)pow(2, -2);utile per trovare il range di rappresentazione*/
            p=&a;
            c=(unsigned char *)p;
            printf("%d %d %d %d\n",*(c+3),*(c+2),*(c+1),*c);
            printf("\n");
            for(i=0;i<4;i++)
              {
                byte[i]=*(c+3-i);
                j=0;
                while (byte[i]>0)
                  {
                bit[j]=byte[i]%2;
                byte[i]= byte[i]/2;
                j=j+1;
              }
            for(;j<8;bit[j++]=0);
            for(j=7;j>=0;j--)
            printf("%d",bit[j]);
            printf(" ");
            }
            printf("\n \n");
            printf("altro numero?\n");
            }
            while(getch()!='n');
}
```