Aritmetica dei Calcolatori 3

Laboratorio di Architettura

31 maggio 2013

Numeri Floating Point (FP)

Il termine Floating Point (virgola mobile) indica la codifica usata per memorizzare i numeri razionali (e per approssimazione, i numeri reali) adottata dai calcolatori.
Dato un certo numero di bit, essi vengono suddivisi in gruppi per memorizzare:

- segno s
- esponente e
- mantissa m

Oggi prenderemo in esame la codifica definita dallo standard IEEE754, che permette di memorizzare numeri FP in singola, doppia o quadrupla precisione, cioè utilizzando 32, 64 o 128 bit.

Un passo indietro

Prendiamo un numero razionale (cioè esprimibile come rapporto tra due numeri interi), possiamo sempre esprimerlo in notazione scientifica normalizzata, cioè nella forma:

$$c\ .\ d_1 d_2 d_3 \cdots * 10^e$$

dove $d_i \in 0, \ldots, 9$ e $c \neq 0$.
Esempi:

$$1.250 * 10^2,\ 3.141,\ 1.000 * 10^{-3}$$

Notazione Binaria FP

Data la base 2, osserviamo che se $c \neq 0$, per forza $c == 1$. Quindi, se operiamo in base binaria, possiamo scrivere:

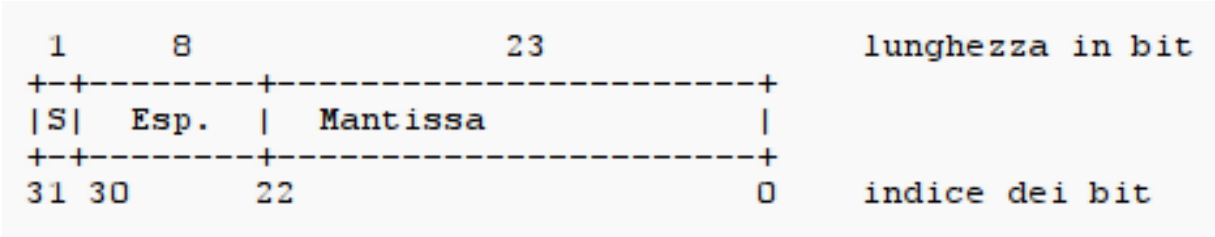
$$. d_1 d_2 d_3 \cdots * 2^e$$

assumendo noto il valore di c.

Standard IEEE754

Lo standard IEEE754 fissa la posizione e dimensione associata ai campi segno, mantissa ed esponente:

float



double

Precisione singola

Abbiamo visto che, per la precisione singola, ritroviamo i seguenti campi:

- segno (bit 31) 1 bit: $0 \rightarrow +, 1 \rightarrow -$
- esponente (bit $23, \ldots, 30$) 8 bit: $e_7\ e_6\ , \ldots, e_0$
- mantissa (bit $0, \ldots, 22$) 23 bit: $m_1\ , m_2 \ldots, m_23$

cioè, considerando 32 bit:

MSB		*LSB*
s	$e_7\ e_6\ e_5\ e_4\ e_3\ e_2\ e_1\ e_0$	$m_1\ m_2\ m_3\ \ldots\ m_{23}$

Ricostruiamo i campi

Mantissa:

$$M = h.m_1\ m_2\ m_3\ \ldots\ m_{23} = h + \sum_{i=1}^{23} m_i * 2^{-i}$$

Esponente:

$$E = \sum_{i=0}^{7} e_i * 2^i$$

Numero Razionale:

$$r = (-1)^s * M * 2^{E-\text{bias}}$$

Per la precisione semplice, bias $= 127$.
Più avanti vedremo il valore da associare a h.

Classificazione dei numeri FP

Considerando i numeri in precisione semplice abbiamo la seguente tabella:

E	M	Categoria	*h*
0	0	zeri	
0	non 0	numeri denormalizzati	0
$1, \ldots, 254$	qualunque	normalizzati	1
255	0	inf	
255	non 0	nan	

Per scoprire a quale categoria appartiene il numero che ci troviamo di fronte, osserviamo, nell'ordine, il valore dell'esponente E e della mantissa M.

Numeri denormalizzati

Abbiamo visto che l'esponente più piccolo è codificato con 1, equivalente a moltiplicare la mantissa per $2^{1-127} = 2^{-126} \simeq 1.17549435 * 10^{-38}$
Questo numero è, in valore assoluto, il numero più piccolo rappresentabile, tra i numeri normalizzati:

$0\ 00000001\ 00000000000000000000000 = +(1.0)_2 * 2^{-126} \simeq 1.17549435 * 10^{-38}$

Per rappresentare numeri il cui valore assoluto è più piccolo, si mette a zero E, in questo caso:

$0\ 00000000\ 10000000000000000000000 = +(0.1)_2 * 2^{-126} \simeq 5.8774717 * 10^{-39}$

Vedi: http://www.ajdesigner.com/fl_ieee_754_word/ieee_32_bit_word.php

Esempi

+0.0000000000000e+00 -> 0x00000000
S=0 E=00000000 (000)M=[0.]00000000000000000000000

-0.0000000000000e+00 -> 0x80000000
S=1 E=00000000 (000)M=[0.]00000000000000000000000

+1.0000000000000e+00 -> 0x3F800000
S=0 E=01111111 (127)M=[1.]00000000000000000000000

+2.0000000000000e+00 -> 0x40000000
S=0 E=10000000 (128)M=[1.]00000000000000000000000

+1.0000000000000e+01 -> 0x41200000
S=0 E=10000010 (130)M=[1.]01000000000000000000000

+1.5000000000000e+00 -> 0x3FC00000
S=0 E=01111111 (127)M=[1.]10000000000000000000000

Esempi

+1.2345600000000e+05 -> 0x47F12000
S=0 E=10001111 (143)M=[1.]11100010010000000000000

+2.1123455047607e+01 -> 0x41A8FCD6
S=0 E=10000011 (131)M=[1.]01010001111110011010110

+inf -> 0x7F800000
S=0 E=11111111 (255)M=[0.]00000000000000000000000

-nan -> 0xFFC00000
S=1 E=11111111 (255)M=[0.]10000000000000000000000

+9.1834094859527e-41 -> 0x0000FFFF
S=0 E=00000000 (000)M=[0.]00000001111111111111111

+1.1754942106924e-38 -> 0x007FFFFF
S=0 E=00000000 (000)M=[0.]11111111111111111111111

Esercitazione (I)

void fp_print(float a)
stampa i valori dei campi S, E, M(aggiungendo il bit h), e le loro rappresentazioni binarie.
Suggerimento:
Come ricavare i bit di a?

- utilizzando memmove:

```
float x; unsigned int y;
x=1.0;
memmove(&y,&x,sizeof(y));
\\a questo punto y vale 0x3F800000
```

- attraverso il cast di un puntatore:

```
float x; unsigned int y;
x=1.0;
y = *((unsigned int *) &x);
\\a questo punto y vale 0x3F800000
```

Esercitazione (II)

void fp_mul(float a, float b, float * c)

1. estrae i campi Sa Ea Ma da a, Sb Eb Mb da b e li memorizza in variabili di tipo unsigned int.
2. controlla la presenza di operandi non validi: se almeno uno tra a e b è Inf o NaN, inizializza a 0.0 il float puntato da c ed esce.
3. reintegra le mantisse: in base a Ea e Eb, setta al valore opportuno il bit 23 di Ma e Mb.
4. calcola il segno del risultato: osserva Sa e Sb e calcola Sc come *Sa* XOR *Sb*
5. calcola il valore temporaneo di $Ec = Ea + Eb - 127$
6. calcola il prodotto tra le mantisse (vedere slide successiva)
7. rinormalizza Mc e modifica Ec nel suo valore finale (vedere slide successiva)
8. compone la stringa di bit floating point posizionando Sc, Ec, Mc nelle posizioni indicate dallo standard (utilizzate una variabile di tipo unsigned int)
9. inizializza al valore calcolato il float puntato da c ed esce.

Prodotto tra mantisse (I)

Dobbiamo calcolare il prodotto tra le mantisse Ma e Mb, memorizzate in un unsigned int (32 bit), dei quali, i 24 bit meno significativi (23 letti dal formato più uno aggiunto in posizione 23 nel punto 3). Abbiamo quindi bisogno di 48 bit, ricavabili da un unisigned long long int (utilizzate cast espliciti!!):

```
unsigned int Ma, Mb;
unsigned long long int temp;
temp = (unsigned long long int) Ma
                * (unsigned long long int) Mb;
```

Ma ora, dov'è la virgola? Come per la moltiplicazione in colonna (quella delle elementari):
$1.1 * 2.3 = 2.53$ si calcola:
$11 * 23 = 253$ e si sposta la virgola dalla posizione a destra verso sinistra di un numero di posizioni pari alla somma del numero di decimali di ogni fattore: $253 \rightarrow 25.3 \rightarrow 2.53$ (2 decimali).

Prodotto tra mantisse (II)

Visto che abbiamo mantisse a 23 cifre dopo la virgola, i bit da prendere da temp saranno dal bit 47 al bit 24 di temp, dovranno essere assegnati alla variabile di tipo unsigned int Mc (nei bit da 23 a 0). Per rinormalizzare la mantissa trovata, osserviamo il bit in posizione 23:

1. la mantissa è normalizzata, mettiamo questo bit a zero (è il bit "nascosto" h) e incrementiamo Ec di uno.
0. la mantissa non è normalizzata: shiftiamo di un posto verso sx Mc e mettiamo il bit 23 a zero per normalizzarla.

Semplificazioni:

- nella nostra simulazione non ci occupiamo di eventuali overflow/underflow di Ec,
- non arrotondiamo Mc, ma semplicemente effettuiamo un trocamento al numero di bit richiesti.

Esempio (I)

```
fp_mul(6.000000e+00,5.000000e+00,&c) = 3.000000e+01
+3.000000000000e+01 -> 0x41F00000
    S=0 E=10000011 (131)M=[1.]11100000000000000000000
+3.000000000000e+01 -> 0x41F00000
    S=0 E=10000011 (131)M=[1.]11100000000000000000000

fp_mul(1.000000e+00,1.175494e-38,&c) = 1.175494e-38
+1.175494210692e-38 -> 0x007FFFFF
    S=0 E=00000000 (  0)M=[0.]11111111111111111111111
+1.175494070562e-38 -> 0x007FFFFE
    S=0 E=00000000 (  0)M=[0.]11111111111111111111110
```

Esempio (II)

l'output di esempio è stato generato da gruppi di chiamate del tipo:

```
a = 6.0f; b = 5.0f;
fp_mul(a,b,&c);
printf("fp_mul(%e,%e,&c)=%e\n",a,b,c);
fp_print(a*b); fp_print(c);
```

Main

Il main deve contenere 10 gruppi di chiamate simili a quelle della slide precedente.

Utilizzate variabili del tipo unsigned int per Ma, Mb, Mc, Sa, Sb, Sc, Ea, Eb, Ec, (unsigned long long int per temp) non utilizzate vettori di interi per memorizzare i bit!

Stampate il valore della variabile floating point, e il valore (in esadecimale) della variabile che avete ottenuto mediante cast (o memmove), dalla quale poi estrarrete (con shift e maschere) i vari campi.

Consegna

La relazione da consegnare è formata da:

1. listato (o listati, se più di uno), in linguaggio C, adeguatamente commentato;
2. uno (o più) file in formato testo contenente la cattura dell'output dei programmi di test. Usate la redirezione verso file:

```
me@mylaptop:~$ ./test > file_output.txt
```

   Oppure copiate l'output e incollate su file di testo. Non usate la cattura di immagini da schermo.

Non includete eseguibili.
Spedite tutto a arc1@fe.infn.it entro le ore 23:59:59 di venerdì 7 giugno.
L'oggetto della mail deve essere nella forma:
LAB1-N#esercitazione-#gruppo
(es: LAB1-N7-99)