Università degli Studi di Cassino e del Lazio Meridionale

Corso di Calcolatori Elettronici

Rappresentazione dei numeri reali

Architettura FP del MIPS

Anno Accademico 2015/2016

Francesco Tortorella

Numeri reali in base 2

- La rappresentazione dei numeri reali in base 2 è completamente analoga a quella in base 10:
  - Parte intera + parte frazionaria, separate da un punto
- La parte frazionaria è formata da cifre che pesano le potenze di 2 a esponente negativo.
  - Esempio: $110.0101_2 \rightarrow 2^{+2}+2^{+1}+2^{-2}+2^{-4} = 6.3125$
- Conversione: si convertono separatamente la parte intera e quella frazionaria.
- Come si converte la parte frazionaria ?

Conversione base 10 → base 2 (frazionari)

Consideriamo un numero F minore di 1.

$$F = c_{-1}x2^{-1} + c_{-2}x2^{-2} + \ldots + c_{-n}x2^{-n} \qquad c_i \in \{0,1\}$$

$$Fx2 = c_{-1} + (c_{-2}x2^{-1} + \ldots + c_{-n}x2^{-(n-1)}) = c_{-1} + P_1 \quad P_1<1$$

$$P_1x2 = c_{-2} + (c_{-3}x2^{-1} + \ldots + c_{-n}x2^{-(n-2)}) = c_{-2} + P_2$$

Conversione base 10 → base 2 (frazionari)

```
void convfrac(float F,int c[],int &n)
{
  float P;
  n=0;P=F;
  do {
     c[n]=(int)(P*2);
     P=P*2-c[n];
     n++;
  } while (P!=0 && n<=NMAX);
}
```

La conversione genera le cifre a partire da quella più significativa

Esempio:

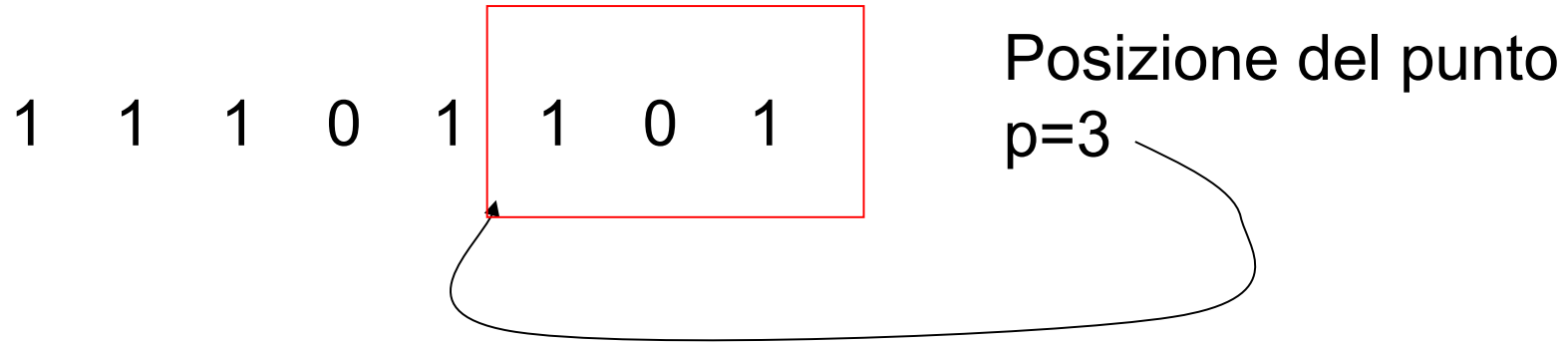
0.625_{10} → **?**$_2$

Rappresentazione nei registri dei numeri reali

- Come rappresentiamo 22.315 ?
- A differenza dei numeri interi, per rappresentare i numeri reali è necessario codificare la posizione del punto frazionario
- Due soluzioni:
  - Codifica esplicita
  - Codifica implicita
- Con la codifica esplicita dovremmo rappresentare sia il numero che il suo fattore di scala → antieconomico e complicato

Rappresentazione in virgola fissa

- Con la codifica implicita, si assume prefissata la posizione del punto all'interno del registro → Rappresentazione in virgola fissa (fixed point)
- Esempio:



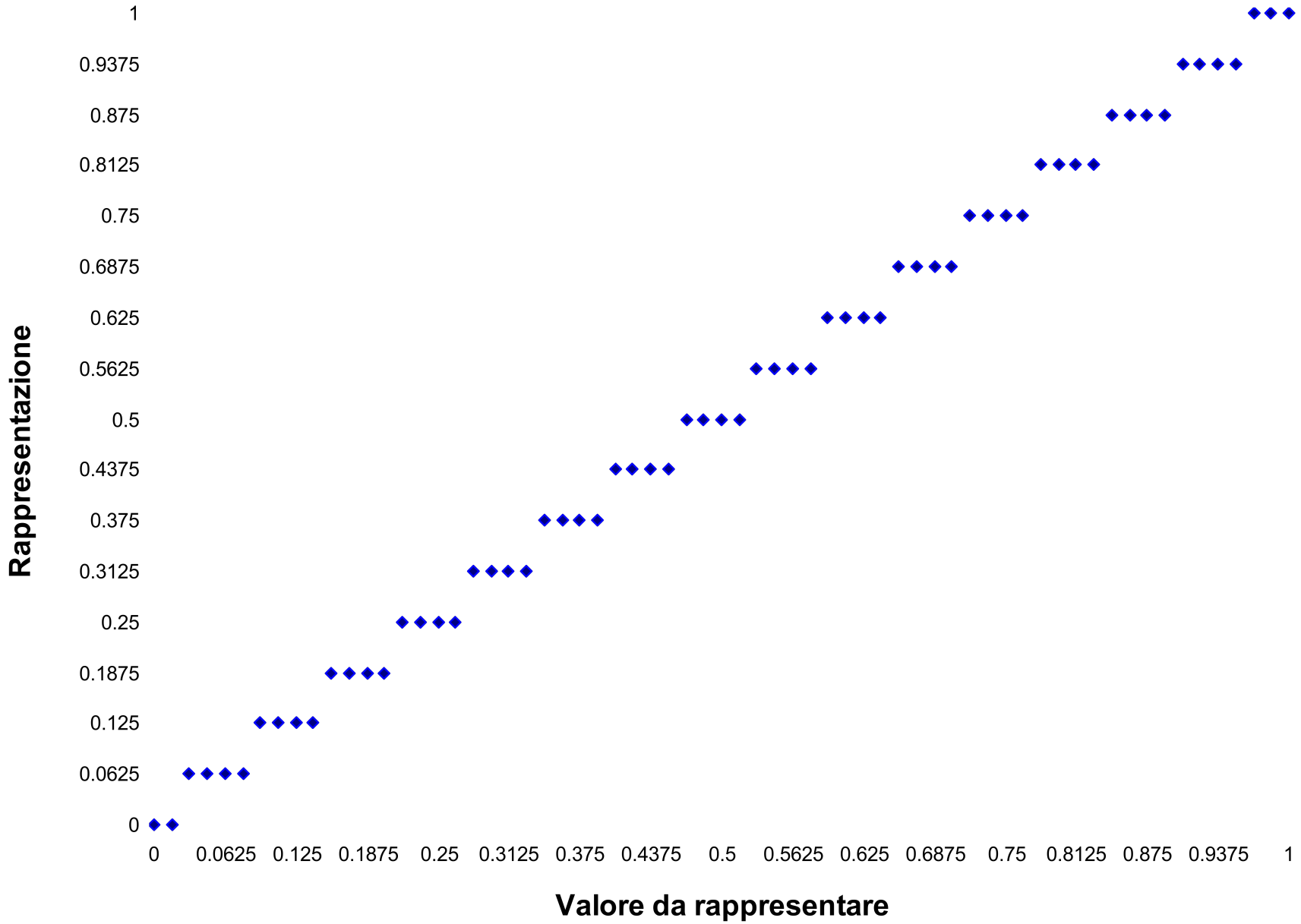
il numero rappresentato è 11101.101

Rappresentazione in virgola fissa

- Con questa convenzione, il valore X rappresentato nel registro è $K*2^{-p}$, dove K è il valore che otterremmo se interpretassimo come un intero il contenuto del registro.
- Qual è l'insieme dei valori rappresentabili su un registro a N bit ?

  K: $0,1,2,\ldots,2^N-1$ → X: $0, 2^{-p}, 2*2^{-p}, \ldots, (2^N-1)*2^{-p}$
- Esempio: N=8, p=4

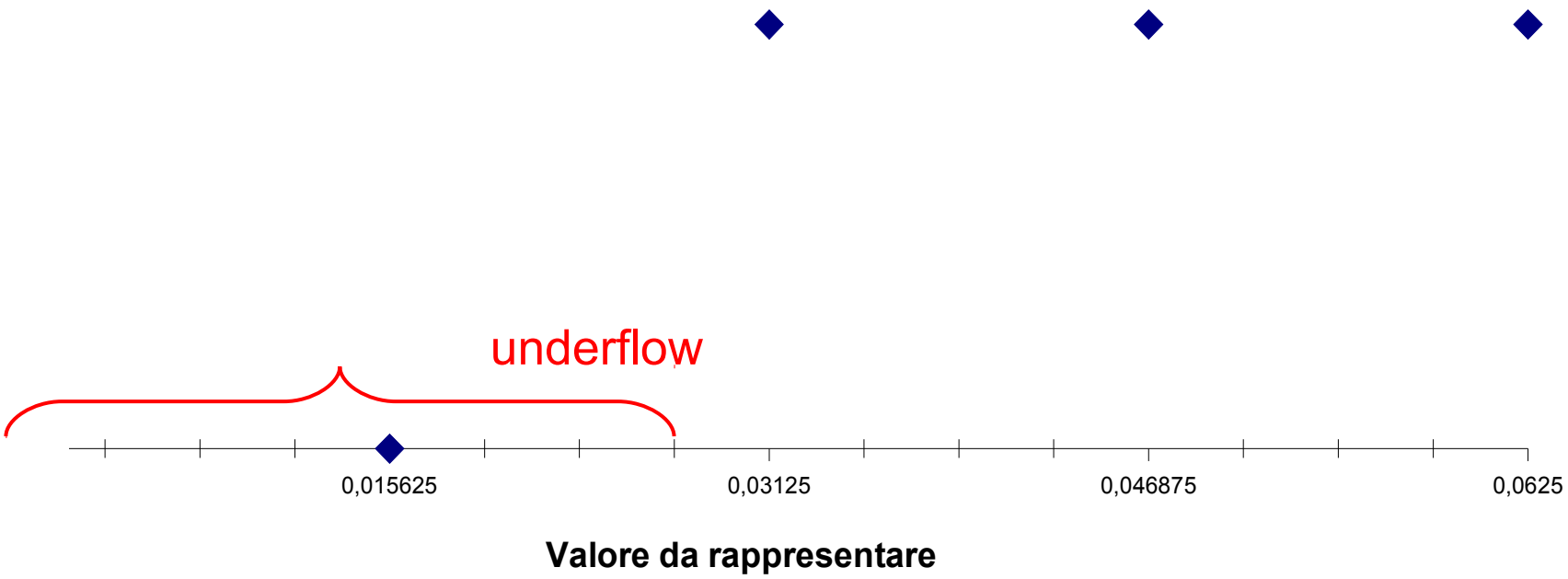
  X= 0, 0.0625, 0.125, 0.1875,…, 15.9375


Rappresentazione
1
0.9375
0.875
0.8125
0.75
0.6875
0.625
0.5625
0.5
0.4375
0.375
0.3125
0.25
0.1875
0.125
0.0625
0
0
0.0625
0.125
0.1875
0.25
0.3125
0.375
0.4375
0.5
0.5625
0.625
0.6875
0.75
0.8125
0.875
0.9375
1
Valore da rappresentare

Rappresentazione in virgola fissa

- I numeri sono rappresentati con una certa approssimazione
  - Esempio: tutti i valori compresi tra 0.03125 e 0.09375 sono rappresentati da 0.0625

- Tutti i valori compresi tra 0 e 0.03125 sono rappresentati da 0.0000 → *underflow*


underflow
0,015625
0,03125
0,046875
0,0625
Valore da rappresentare

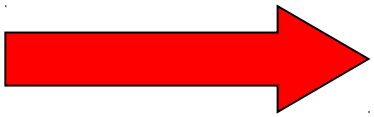
Rappresentazione di un numero in virgola fissa

Supponiamo di voler rappresentare il numero 22.315 in virgola fissa in un registro ad 8 bit con p=3.

Separiamo parte intera e parte frazionaria:

$22_{10} \rightarrow 10110_2$

$0.315_{10} \rightarrow 0.010100\ldots_2$



1 0 1 1 0 0 1 0

Precisione della virgola fissa

- Quantifichiamo l'errore assoluto:

  $Err_{max} = 2^{-p}/2$ → per p=4 $Err_{max}= 0.03125$

- Come fare per diminuire l'errore ?

  basta aumentare p, ma qual è l'effetto sul range dei numeri rappresentabili ?

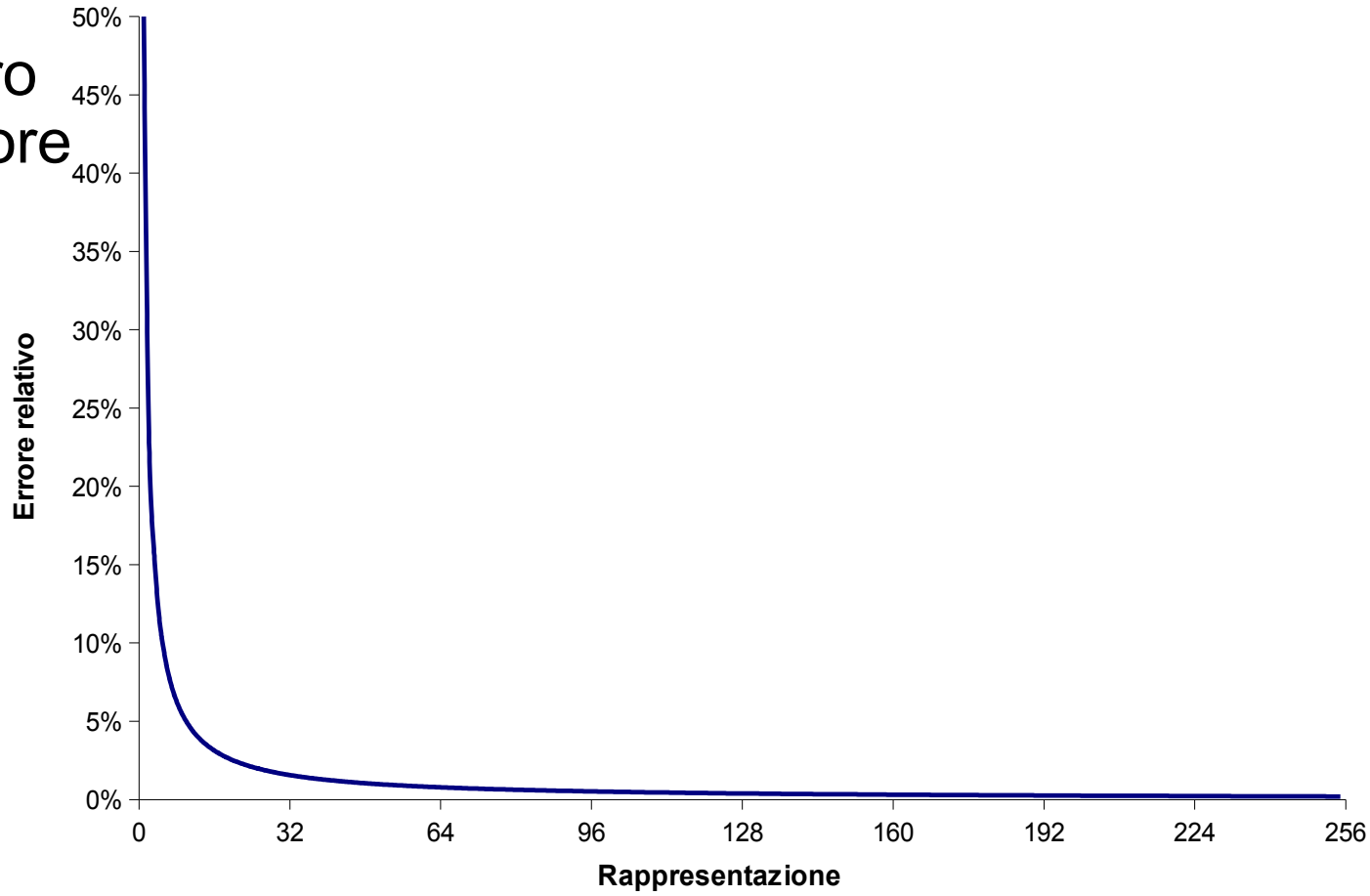
  → compromesso tra range e precisione

- Ricordiamo che X: $0, 2^{-p}, 2*2^{-p}, \ldots, (2^N-1)*2^{-p}$

Precisione della virgola fissa

Il problema vero è legato all'errore relativo:

$E_{rel}=Err_{max}/x$



Virgola fissa con segno

- La codifica dei numeri relativi in complementi alla base si applica in maniera immediata ai numeri reali rappresentati in virgola fissa.
- La rappresentazione di un numero reale con segno (N bit, punto in posizione p) si ottiene tramite la regola:

$$\begin{cases} R(x) \text{ se } x \geq 0 \\ b^{N-p} - R(|x|) \text{ se } x < 0 \end{cases}$$

dove R(x) è la rappr. in virgola fissa di x

Virgola fissa con segno

- In questo modo, l'intervallo dei numeri rappresentabili diventa:

$$[(-2^{N-1})*2^{-p} \quad (+2^{N-1}-1)*2^{-p}]$$

  oppure:

$$[-2^{N-p-1} \quad +2^{N-p-1}-2^{-p}]$$

- Esempio (N=8, p=3):

$$[-2^{8-3-1} \quad +2^{8-3-1}-2^{-3}]$$

$$[-16.000 \quad +15.875]$$

Virgola fissa con segno

Esempio (N=8, p=3):

R(-3.7) = 2^5-R(3.7) →

	1	0	0	0	0	0	0	0	0	-
		0	0	0	1	1	1	0	1	
R(-3.7) →	1	1	1	0	0	0	1	1		

Virgola fissa con segno

- Possiamo comunque applicare il criterio già visto per ottenere velocemente la rappresentazione in complementi alla base:
- Per ottenere R(-3.7) si considera R(3.7) e si complementa cifra per cifra aggiungendo 1 al bit meno significativo:

R(3.7) 0 0 0 1 1 1 0 1

1 1 1 0 0 0 1 0 +

1

1 1 1 0 0 0 1 1

Riassumendo…

- La rappresentazione in virgola fissa ha innegabili vantaggi:
  - Semplicità
  - Piena compatibilità con la rappresentazione degli interi e possibilità di usare circuiti aritmetici comuni.
- Ma ha anche grossi problemi:
  - Errore relativo elevato per x →0
  - Compromesso range/precisione
  - Entrambi legati al fatto che il fattore di scala è fisso.

La virgola è mobile…

- Si potrebbero mitigare i problemi andando a rappresentare esplicitamente il fattore di scala.
- In questo modo la virgola non è più “fissa”, ma diventa “mobile”.
- Rappresentazione in virgola fissa → Rappresentazione in virgola mobile (floating point)

Rappresentazione in virgola mobile

- Fissata la base b, il valore viene considerato nella forma $M*b^E$ (notazione scientifica) ed è rappresentato tramite la coppia (M,E)

  Esempio: $22.315=0.22315*10^2$ → (0.22315,2)

  $10110.010=10.110010*2^3$ → (10.110010,11)

- Nel registro saranno quindi prefissate zone diverse per la mantissa e per l'esponente

Rappresentazione in virgola mobile

Come si rappresentano M ed E ?

- M
  - numero reale
  - segno e modulo
  - virgola fissa
- E
  - numero intero con segno
  - eccessi
- La disposizione nel registro facilita il confronto

Intervallo di numeri rappresentabili

- M rappresentato su m bit con p cifra frazionarie
  M: $0, 2^{-p}, 2*2^{-p}, \ldots, (2^{m}-1)*2^{-p}$

- E rappresentato su e bit
  E: $-2^{e-1},\ldots,+2^{e-1}-1$

- $N_{min}=M_{min}*2^{Emin} = 2^{-p} * 2^{(-2e-1)}$

- $N_{max}=M_{max}*2^{Emax} = (2^{m}-1)* 2^{-p}*2^{(+2e-1}-1)$

Intervallo di numeri rappresentabili

- Esempio:
  - m=23 p=23
  - e=8
- $N_{min} = 2^{-23} * 2^{-128} \cong 3.5 * 10^{-46}$
- $N_{max} = (2^{23}-1) * 2^{-23} * 2^{127} \cong 1.7 * 10^{+38}$

Esempio

Rappresentazione in FP di –12.6:

$12.6_{10} = 1100.\overline{1001}_2 = 0.11001\overline{001} * 2^4$

Segno: 1

Mantissa: 0.11001001100110011001100

Esponente: $4+128 = 132_{10} = 10000100_2$

Rappresentazione normalizzata

- Con la virgola mobile non c'è unicità di rappresentazione:
  $N = M*2^E = (M*2)*2^{E-1} = (M*4)*2^{E-2} = (M/2)*2^{E+1}$

- Quale scegliere ? Quella che massimizza la precisione:
  prima cifra della mantissa diversa da 0
  → *rappresentazione normalizzata*

Rappresentazione normalizzata

- Esempio: N = 0.0003241892
  mantissa a 5 cifre decimali
- Diverse rappresentazioni possibili:

  $0.00032*10^{0}$

  $0.00324*10^{-1}$

  $0.03241*10^{-2}$

  $0.32418*10^{-3}$ ← normalizzata

Rappresentazione normalizzata

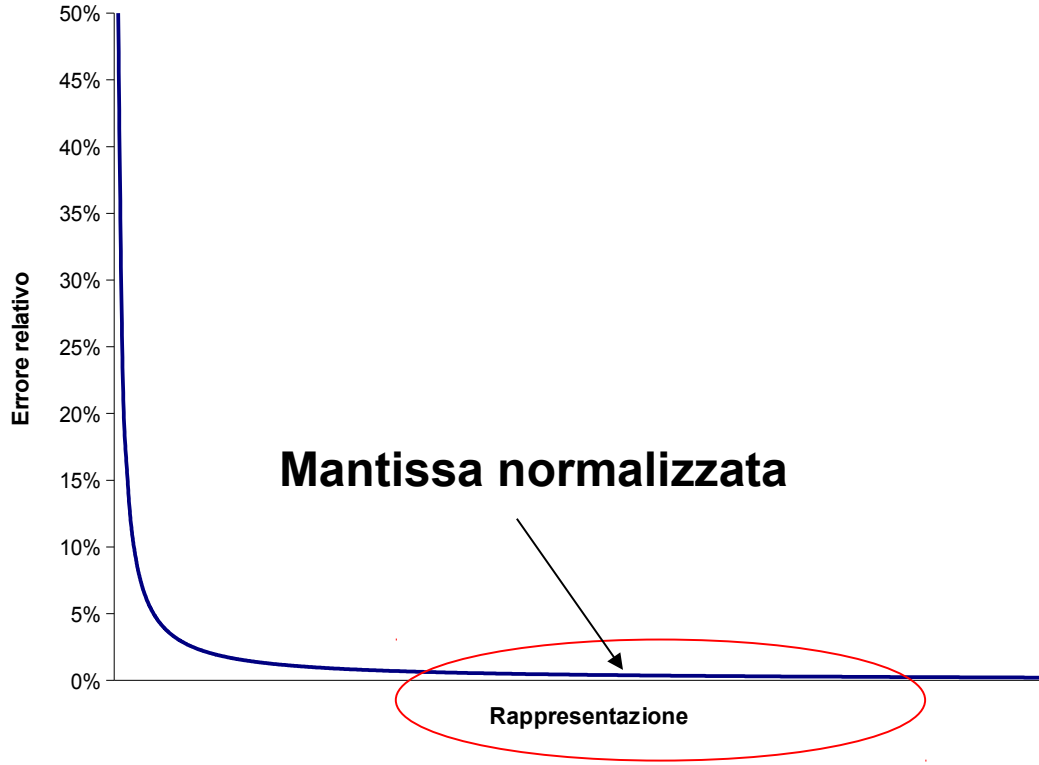
- L'intervallo di rappresentazione si modifica :
  $N_{min} = 2^{m-1} * 2^{-p} * 2^{-2^{e-1}}$

- Esempio:
  - m=23 p=23
  - e=8

- $N_{min} = 2^{-23} * 2^{-128} \cong 3.5 * 10^{-46}$ (non normalizzata)
- $N_{min} = 2^{22} * 2^{-23} * 2^{-128} \cong 1.5 * 10^{-39}$ (normalizzata)

Rappresentazione normalizzata

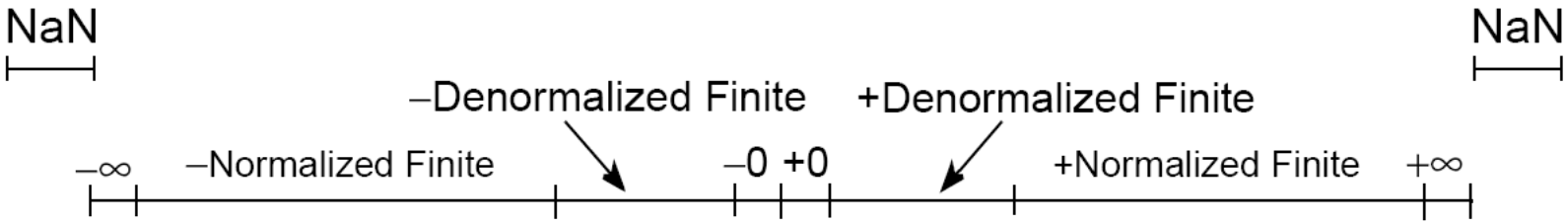
- Valutiamo l'errore di approssimazione:
  - Errore assoluto massimo: $Err_{max} = (2^{-p}/2)*2^{E}$
  - Errore relativo: $E_{rel} = Err_{max}/x$
- Pro
  - Maggiore precisione
- Contro
  - Underflow più frequente



Lo standard IEEE754

- Due formati
  - 32 bit: 23 bit mantissa + 8 bit esp. + 1 bit segno, bias=127
  - 64 bit: 52 bit mantissa + 11 bit esp. + 1 bit segno, bias=1023
- Mantissa con *hidden bit*
  $N = (-1)^s * (1.M) * 2^{E-bias}$
- Esponente polarizzato
  - Sono riservate le rappresentazioni dell'esponente 00…0 e 11…1
- Underflow graduale, denormalizzazione

Lo standard IEEE754



Real Number and NaN Encodings For 32-Bit Floating-Point Format

S	E	F			S	E	F
1	0	0	–0	+0	0	0	0
1	0	$0.XXX^2$	–Denormalized Finite	+Denormalized Finite	0	0	$0.XXX^2$
1	1...254	Any Value	–Normalized Finite	+Normalized Finite	0	1...254	Any Value
1	255	0	–∞	+∞	0	255	0
X^1	255	$1.0XX^2$	–SNaN	+SNaN	X^1	255	$1.0XX^2$
X^1	255	1.1XX	–QNaN	+QNaN	X^1	255	1.1XX

NOTES:
1. Sign bit ignored.
2. Fractions must be non-zero.

Lo standard IEEE754

	Range denormalizzato	Range normalizzato	Decimale
32 bit	Min: 2^{-149} Max:$(1-2^{-23})\times2^{-126}$	Min: 2^{-126} Max: $(2-2^{-23})\times2^{127}$	1.4×10^{-45} 3.4×10^{38}
64 bit	Min: 2^{-1074} Max:$(1-2^{-52})\times2^{-1022}$	Min: 2^{-1022} Max: $(2-2^{-52})\times2^{1023}$	4.9×10^{-324} 1.8×10^{308}

Lo standard IEEE 754

- Esistono rappresentazioni riservate (definite "numeri speciali") che permettono l'estensione dell'aritmetica a casi particolari:
  - NaN (0/0, sqrt(-2^k))
  - $+\infty$, $-\infty$
  - **denormalizzato** ➡

E	M	N
255	**≠0**	**NaN**
255	**=0**	**$(-1)^s \infty$**
1-254	**qualunque**	+/- numero fp
0	**0**	**0**
0	**≠0**	**$(-1)^s * 2^{-126} * (0.M)$**

Lo standard IEEE 754

Operazioni speciali:

Le operazioni sui numeri speciali sono ben definite dalla standard IEEE. Nel caso più semplice, ogni operazione con NaN fornisce come risultato NaN.

Le altre operazioni sono definite come in tabella.

OPERAZIONE	RISULTATO
n / ±Inf	0
±Inf × ±Inf	±Inf
± nonzero / 0	±Inf
Inf + Inf	Inf
±0 / ±0	NaN
Inf - Inf	NaN
±Inf / ±Inf	NaN
±Inf × 0	NaN

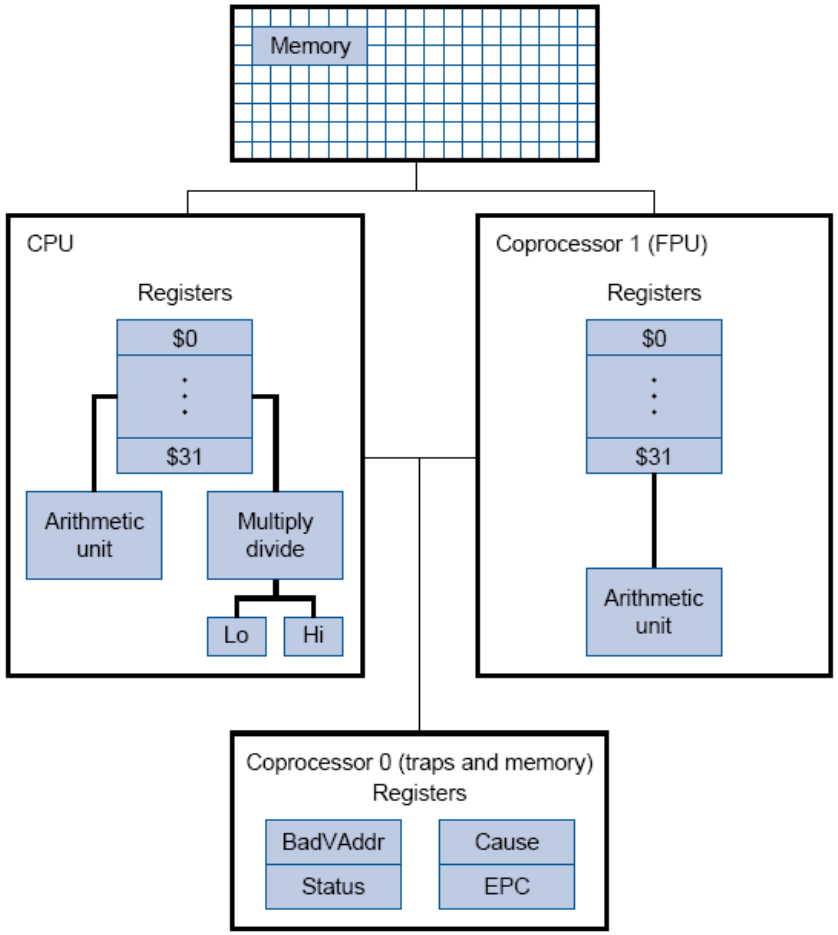
Addizione e sottrazione in floating point

- Molto più complicate rispetto agli interi e alla virgola fissa
- Diverse operazioni necessarie:
  - Denormalizzazione per allineare i valori all'esponente più alto
  - Sommare le mantisse
  - Normalizzare il risultato e verificare se si è in under/overflow
  - Arrotondare se necessario (può richiedere un'ulteriore normalizzazione)
- Se i segni sono diversi, bisogna calcolare la differenza tra le mantisse e determinare il segno del risultato
- Operazioni troppo complesse per poter essere effettuate con l'unità aritmetica per gli interi.

L'architettura floating point del MIPS

- Il MIPS non ha all'interno della CPU l'hardware FP, ma impiega un coprocessore dedicato (coprocessore 1)
- In questo modo si evita un'eccessiva complessità della CPU.
- I processori attuali integrano una o più FPU (Floating Point Unit) indipendenti.

L'architettura floating point del MIPS



L'architettura floating point del MIPS

- Il coprocessore 1 ha 32 registri da 32 bit (singola precisione): **`$f0, $f1, …, $f31`**
- E' possibile gestire dati in doppia precisione (64 bit) usando coppie di registri consecutivi pari-dispari (es. `$f0-$f1`, `$f2-$f3`, …) come fossero registri da 64 bit. Il nome del registro risultante coincide con quello del registro pari.
- Classi di istruzioni:
  - Aritmetiche (singola e doppia precisione)
  - Trasferimento dati memoria/coproc. e coproc./memoria
  - Salto condizionato

Istruzioni aritmetiche

`add.s $f2,$f3,$f4`	$f2=$f3+$f4
`sub.s $f2,$f3,$f4`	$f2=$f3-$f4
`mul.s $f2,$f3,$f4`	$f2=$f3*$f4
`div.s $f2,$f3,$f4`	$f2=$f3/$f4

Singola precisione

`add.d $f2,$f4,$f6`	$f2=$f4+$f6
`sub.d $f2,$f4,$f6`	$f2=$f4-$f6
`mul.d $f2,$f4,$f6`	$f2=$f4*$f6
`div.d $f2,$f4,$f6`	$f2=$f4/$f6

Doppia precisione

Istruzioni per il trasferimento dati

Coprocessore ←→ memoria

`lwc1 $f1,100($t2)`	$f1 ← Mem[$t2+100]$_{32}$
`swc1 $f3,150($t3)`	Mem[$t3+150]$_{32}$ ← $f3
`ldc1 $f2,100($t2)`	[$f2:$f3] ← Mem[$t2+100]$_{64}$
`sdc1 $f4,150($t3)`	Mem[$t3+150]$_{64}$ ← [$f4:$f5]

Coprocessore ←→ CPU

`mtc1 $t2,$f3`	$f3 ← $t2
`mfc1 $t3,$f5`	$t3 ← $f5

Trasferimento dati Coprocessore ←→ CPU

- Bisogna tenere presente che le rappresentazioni dei dati tra registri della CPU e registri del coprocessore sono diverse (interi e complementi alla base vs. IEEE754)
- Il trasferimento del contenuto di un registro della CPU in un registro del coprocessore opera soltanto una copia della stringa di bit, non realizza una conversione

Trasferimento dati Coprocessore ←→ CPU

- Per realizzare una conversione int → float esiste l'istruzione **cvt.s.w fd,fs** che converte il valore intero presente nel registro **fs** in un float, trasferendolo poi nel registro **fd**.

- **Esempio**: Per azzerare $f0 ($f0 <- 0.0)

```
mtc1 $zero,$f0  # trasferisce in f0 il valore 0(int)
cvt.s.w $f0,$f0 # conversione int -> float(single)
```

- **Esempio**: per trasferire nel registro $f2 un valore intero presente in memoria all'etichetta val

```
lwc1 $f1,val    # trasferisce in f1 il valore (int)
cvt.s.w $f2,$f1  # conversione int -> float(single)
```

Istruzioni di conversione

cvt.*to*.*from* $fd,$fs

Converte il valore presente in $fs e lo trasferisce in $fd

***to* e *from* possono assumere i seguenti valori**

s single

d double

w integer

Istruzioni di salto condizionato

- Il coprocessore utilizza un flag (condition code) per gestire i salti condizionati.
- Il flag viene posto a 0 o a 1 dalle istruzioni di confronto e viene consultato dalle istruzioni di salto condizionato.

Istruzioni di salto condizionato

Singola precisione

`c.xy.s $f1,$f3` if $f1 (xy) $f3 cond=1 else cond=0

`xy` indica uno degli operatori di confronto:

`eq, ne, lt, le, gt, ge`

Doppia precisione

`c.xy.d $f2,$f6` if [$f2:$f3] (xy) [$f6:$f7] cond=1 else cond=0

`xy` indica uno degli operatori di confronto:

`eq, ne, lt, le, gt, ge`

Istruzioni di salto condizionato

`bc1t offset` if cond==1 goto PC+4+offset

`bc1f offset` if cond==0 goto PC+4+offset

- Come al solito, nel programma assembly l'operando sarà un'etichetta di istruzione in base alla quale l'assemblatore calcolerà l'offset.

esempio

```
        .data

A:      .float 7.4
B:      .float 3.1
C:      .space 4

        .text
main:   la   $t0,   A              # in $t0 l'indirizzo di A
        la   $t1,   B              # in $t1 l'indirizzo di B
        la   $t2,   C              # in $t2 l'indirizzo di C

        lwc1   $f1, 0($t0)         # carica il valore in A
        lwc1   $f2, 0($t1)         # carica il valore in B
        add.s $f3, $f1, $f2        # esegue l'operazione fp
        swc1   $f3, 0($t2)         # memorizza il risultato in C

        li $v0,10
        syscall
```