UNION CAST/CAMPI DI BIT

martedì 20 dicembre 2022 15:13

UNION CAST

INT x = 3.2F;
↳ x vale 3. (TRONCA)

3,2 È codificabile IN BINARIO e a me INTERESSA accedere a quella SPECIFICA RAPPRESENTAZIONE BINARIA;
Voglio INTERPRETARE quella STRINGA di BIT IN un altro modo, ma senza effettuare la conversione a TIPO;

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

union binary_float_t
{
    float real;
    uint32_t integer; // Assumes float is 32 bits wide };

};


int main(void)
{
    union binary_float_t f;
    f.real = 3.141592F;
    printf("Hex representation of %f is %#04x\n", f.real, f.integer);
    return 0;
}
```

Passo a convertire un valore da un tipo all'altro!

è la x sulla formattazione che fa la differenza

Esadecimale 04, 4 cifre decimali

0x esadecimale

#

OUTPUT:

```
Hex representation of 3.141592 is 0x40490fd8
```

Scrivo sulla memoria UNION un valore con una certa RAPPRESENTAZIONE, E posso RILEGGERE quel valore secondo un'altra RAPPRESENTAZIONE.
Questo perché la UNION condivide una sola area di memoria per tutte le variabili che ne fanno parte;

IL TIPO PIÙ PICCOLO CHE POSSIAMO GESTIRE È IL CHAR!
PERÒ qualche volta mi potrebbe INTERESSARE lavorare alla grana del singolo bit!
Questa cosa viene fatta con i cosiddetti **CAMPI DI BIT!**
IO NON POSSO INDIRIZZARE nulla che sia più piccolo di 1 BYTE, PERÒ AD esempio VORREI GESTIRE singoli BIT;
Immaginiamo di SCRIVERE un PROGRAMMA PER GESTIRE singolarmente IL REGISTRO flags, PER esempio.
Quello che posso fare è gestire i singoli campi di BIT per andare a prendere all'INTERNO di una STRUCT, singoli BIT o gruppi di singoli BIT.
PER esempio VORREI POTER UTILIZZARE 8 variabili BOOLEANE all'INTERNO di un char, PER lavorare alla grana del singolo BIT;

LO POSSO FARE COSÌ:

```c
union flt {
        struct ieee754 {
                uint32_t mantissa: 23;
                uint32_t exponent: 8;
                uint32_t sign: 1;
        } raw;
        float f;
};
```

questa STRUCT ha 3 membri: UINT32 bit, come nel caso precedente!
CON la SCRITTURA **:N**, Dico che mi SERVE UN membro che sia grande N BIT;
IL PRIMO membro 23 BIT, IL secondo 8 BIT, IL TERZO 1.

ABBIAMO effettuato la seguente suddivisione:

IEEE-754

QUESTO È UNO STANDARD A 32 BIT E SI AVVALE DEL SEGUENTE SCHEMA:

IEEE-754

QUESTO È UNO STANDARD A 32 BIT, E SI AVVALE DEL SEGUENTE SCHEMA:

QUI SI LAVORA ALLA GRANA DEL SINGOLO BIT!

23+8+1 = 32 e questi 32 bit vengono messi tutti nello stesso UINT32. sono contigui!

MA QUESTA STRUCT, CON i CAMPI DI BIT, DIVENTA UN MEMBRO DI UNA UNION.

A questo punto posso accedere ai singoli bit e settarli come voglio:

```
number.raw.sign = 1;
number.raw.exponent = 120;
number.raw.mantissa = 1685475;

printf("\fConverting %d %s %s to float:\n", number.raw.sign, exponent,
       mantissa);
printf("\t%f\n", number.f);
```

CODICE COMPLETO

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

union flt
{
    struct ieee754
    {
        uint32_t mantissa: 23;
        uint32_t exponent: 8;
        uint32_t sign: 1;
    } raw;
    float f;
} number;


int main(void)
{
    number.raw.sign = 1;
    number.raw.exponent = 120;
    number.raw.mantissa = 1685475;
    printf("\fConverting %d %d %d to float:\n", number.raw.sign, number.raw.exponent, number.raw.mantissa);
    printf("\t%f\n", number.f);
    return 0;
}
```

] setting

OUTPUT: → 1 negativo

```
Converting 1 120 1685475 to float:
        -0.009382
```

LE ISTRUZIONI ASSEMBLY NON GESTISCONO IL "BYTE" come TIPO più piccolo! come è possibile che io possa scrivere un singolo bit?

IN REALTÀ QUESTE OPERAZIONI QUI NON SCRIVONO SINGOLI BIT, PERCHÈ SCRIVERE SINGOLI BIT NON È POSSIBILE!
VENGONO EMULATE CON delle **operazioni logiche**!

QUESTE operazioni effettuano calcoli al livello del

QUESTE OPERAZIONI EFFETTUANO CALCOLI AL LIVELLO DEL SINGOLO BIT!

MASCHERE DI BIT

PROVIAMO A FORZARE UN BIT A UNO! OR (+)

Voglio che sia 1 → 0111 + 1111 = 1111

```
orl $0x80000000, %eax
```

PROVIAMO A FORZARE UN BIT A ZERO! AND (·)

1111
0111
0111

```
andl $0x7FFFFFFF, %eax
```

POSSO INVERTIRE UN BIT CON LO XOR (XOR)

1000
1000
0000

```
xorl $0x80000000, %eax
```

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

SE ESEGUO LO XOR FRA UN REGISTRO E SE STESSO, AZZERO I REGISTRI:

- Per azzerare un registro, si possono usare due istruzioni equivalenti:
  - `movq $0, %rax`
  - `xorq %rax, %rax`
- La seconda è preferibile perché più efficiente

LE MASCHERE DI BIT FANNO SÌ CHE LAVORINO IN BASE AL NUMERO CHE DEVO RAPPRESENTARE, E QUINDI SI MODIFICANO MOLTO SPESSO;

PERÒ A ME INTERESSA ANCHE ESTRARRE DEI BIT, PER LEGGERLI!

SE voglio vedere che almeno uno dei 3 bit meno significativi è diverso da zero, creo una maschera: 000...0111 (32 bit)

=

$(7)_{10}$

ORA: Supponiamo di avere un numero a 32 bit e di voler estrarre il valore dei 3 bit meno significativi

SI EFFETTUA L'AND BIT A BIT

AND:

```
10101010110110101011110000001111
00000000000000000000000000000111
--------------------------------
00000111
```

Estrazione dei bit: `andl $7, %eax`

TEST, È UN'ISTRUZIONE CHE SERVE PER EFFETTUARE DEI CONTROLLI LOGICI, CALCOLA L'AND E SCARTA IL RISULTATO;

Per verificare se i bit sono a zero: `testl $7, %eax`

Che cosa fa l'istruzione `testq %rax, %rax`?

AND LOGICO TRA UN VALORE E SE STESSO!

SE RAX≠0, IL RISULTATO È ≠0, E $Z_F = 0$!
SE RAX=0, IL RISULTATO È =0, E $Z_F = 1$!

IF (X==0) ?