C / ASSEMBLY: CONFRONTO

mercoledì 30 novembre 2022 16:42

RELAZIONE TRA C E ASSEMBLY.

C

```c
#include <stdio.h>

/* This is a comment. */
int main(int argc, char *argv[])
{
    int times = 100;

    // this is also a comment
    printf("Hello World! I
      welcome you %d times.\n",
      times);
    return 0;
}
```

ASS.

```
.data
.LC0:
        .ascii  "Hello World! I welcome
                 you %d times.\n"

.text
main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $8, %rsp
        movl    $100, -4(%rbp)
        movl    -4(%rbp), %esi
        leaq    .LC0(%rip), %rdi
        xorl    %eax, %eax
        call    printf@PLT
        xorl    %eax, %eax
        addq    $8, %rsp
        leave
        ret
```

→ all' INTERNO della sezione data AVRÒ un insieme di BYTE che mi rappresenta quella STRINGA.

Questa STRINGA è una variabile globale che è definita a Tempo di compilazione che va nella sezione data, essendo diversa da 0!
PERCHÈ è una variabile globale diversa da 0.
NON è locale a NESSUNA funzione.

IL MIO COMPILATORE emetterà una direttiva verso l'assemblatore, .data, che significa che da questo momento in poi TUTTO quello che SCRIVO deve andare a finire nella sezione .data;

SUBITO dopo leggo .ASCII, ossia una direttiva che mi dice che sto dichiarando una variabile che è una STRINGA;

Quindi da qualche parte IN MEMORIA TROVERÒ la scritta di hello world. È una STRINGA. Questa STRINGA, che è un parametro della funzione di PRINTF, deve necessariamente diventare una variabile globale.

Devo appuntarmi l'INDIRIZZO IN MEMORIA IN cui questa STRINGA INIZIA. INTRODUCIAMO UNA LABEL. $LC0: È una label.

INIZIA col . quando ci RIFERIAMO a posizioni INTERNE NEL codice, senza . quando parliamo di funzioni;

IL valore di questa etichetta È L'INDIRIZZO DELL' oggetto IMMEDIATAMENTE SUCCESSIVO;

È un etichetta che viene aggiunta dall' ASSEMBLATORE all' INTERNO del mio PROGRAMMA assembly.

SUBITO dopo questa ETICHETTA c'è la DICHIARAZIONE della mia STRINGA;
LC0 avrà l'INDIRIZZO di quella STRINGA di MEMORIA SUCCESSIVA;

TUTTO IL RESTO FINISCE ALL'INTERNO DELLA SEZIONE .TEXT perché ho delle ISTRUZIONI che devono essere CONVERTITE IN ISTRUZIONI MACCHINA.

Dopo TEXT TROVO un'altra label, ossia **MAIN**: NON COMINCIA CON IL PUNTO, quindi È ASSOCIATA AD UNA FUNZIONE;
RICHIAMA ESATTAMENTE IL NOME della funzione di alto livello IN C.

PER OGNI FUNZIONE nel mio programma assembly ci sarà una LABEL.
Qual è il valore di INDIRIZZO CHE HA questa label MAIN? L'INDIRIZZO della PRIMA ISTRUZIONE. LA FUNZIONE COMINCIA CON la PRIMA ISTRUZIONE ASSEMBLY.
PRIMA l'oggetto immediatamente successivo era un dato in memoria, qui è un'ISTRUZIONE ASSEMBLY.
MAIN VARRÀ l'INDIRIZZO di memoria dell'ISTRUZIONE immediatamente successiva, ossia la pushq %rbp.

ANALIZZIAMO IL MAIN:

```
                                    .data
                                    .LC0:
1. #include <stdio.h>                    .ascii  "Hello World! I welcome
2.                                                  you %d times.\n"
3. /* This is a comment. */
4. int main(int argc, char *argv[])     .text
5. {                                    main:
6.     int times = 100;                     pushq   %rbp
7.                                          movq    %rsp, %rbp
8.     // this is also a comment            subq    $8, %rsp
9.     printf("Hello World! I               movl    $100, -4(%rbp)
         welcome you %d times.\n",          movl    -4(%rbp), %esi
         times);                            leaq    .LC0(%rip), %rdi
10.    return 0;                            xorl    %eax, %eax
11. }                                       call    printf@PLT
                                            xorl    %eax, %eax
                                            addq    $8, %rsp
                                            leave
                                            ret
```

OP. IN PREP.

La dichiarazione della funzione comincia sempre con un PREAMBOLO.
È una serie di operazioni che operano sullo STACK per andare a creare un'area di memoria in cui la funzione può memorizzare delle variabili.

POSTAMBOLO

CORPO della FUNZIONE:

La funzione deve scrivere 100 alla sua variabile locale e la scriviamo sullo STACK, nell'area di memoria che abbiamo preparato precedentemente.
LO STACK PERMETTE di mantenere i valori delle variabili locali alla funzione.

POI VA CHIAMATA LA PRINTF(), E DOBBIAMO PASSARE I NOSTRI PARAMETRI alla FUNZIONE;
IL PRIMO PARAMETRO DELLA NOSTRA FUNZIONE È LA STRINGA, come la passo?
GLI PASSO L'INDIRIZZO: LEA, serve a calcolare gli indirizzi e non accedere in memoria, così vado a scrivere all'interno di %rdi, un registro calcolato!
È UNA modalità di INDIRIZZI descritta come spiazzamento a partire dal RIP.
"SCRIVI dentro RDI, uno spiazzamento a partire dal RIP, che considera registro di BASE, per raggiungere l'INDIRIZZO calcolato, che è l'INDIRIZZO associato alla mia variabile globale".

IL secondo parametro della mia funzione è times, È un INTERO e RIESCE ad ENTRARE IN UN REGISTRO, quindi IN questo caso posso NON PASSARGLI un INDIRIZZO ma ESATTAMENTE IL VALORE, MI VADO A vedere dov'era questa variabile e l'avevo scritto

IN UN REGISTRO, quindi IN questo caso posso NON PASSARGLI un indirizzo ma ESATTAMENTE il valore, MI VADO A vedere dov'era questa variabile e l'accesso SCRIVO a questo INDIRIZZO di memoria, UTILIZZO quell'INDIRIZZO di memoria come sorgente, e copio il contenuto di quell'area di MEMORIA IN UN altro REGISTRO;

(%esi)

A questo PUNTO ho caricato IN due REGISTRI l'indirizzo della mia STRINGA e il valore che la PRINTF deve stampare.

La caratteristica di queste funzioni che accettano un numero qualsiasi di Parametri è di specificare quanti valori IN virgola mobile stiamo passando:

XORL %EAX, %EAX

NON NE PASSIAMO NESSUNO e specifichiamo che passiamo 0 PARAMETRI.
LO XOR fra due variabili BOOLEANE identiche è 0.
Si AZZERA IL CONTENUTO del REGISTRO;

A questo PUNTO abbiamo preparato i parametri, ora trasferiamo il CONTROLLO alla funzione PRINTF();

CALL PRINTF@PLT

È un modo IN CUI il SISTEMA gestisce le funzioni di LIBRERIA:

ossia la PRINTF è una funzione di libreria; TROVATELA 😊

E così stampiamo il messaggio:

Ora la funzione deve RITORNARE RETURN 0; se è tutto apposto!
Quindi per ogni funzione avremo il cosiddetto POST-AMBOLO!

XORL %EAX, %EAX

AZZERIAMO UN REGISTRO;

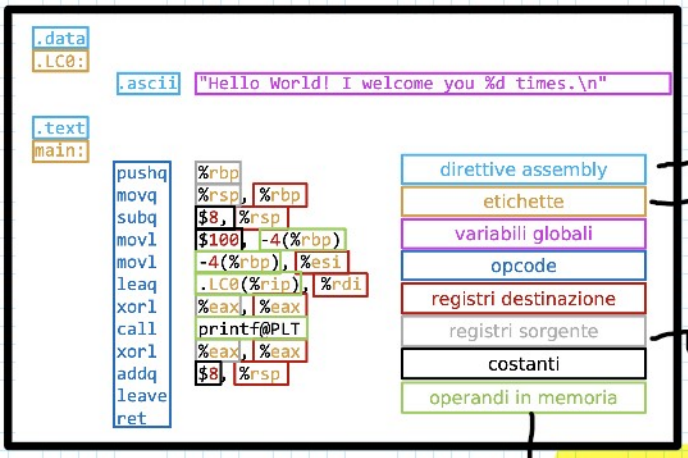
Questo sarà il valore di RITORNO della mia funzione;

Rimettiamo lo STACK a POSTO, esattamente come l'abbiamo TROVATO dopo che la funzione ha preso il CONTROLLO;

Ogni funzione viene quindi logicamente divisa IN 3 fasi;

Un PREAMBOLO in cui mi preparo le mie aree di memoria per andare a supportare l'esecuzione della mia funzione.
IL CORPO della funzione.
UN POST-AMBOLO per RESTITUIRE al chiamante con la funzione RETURN;

SPEZZIAMO IL PROGRAMMA:



→ NON compare nella RAPPRESENTAZIONE BINARIA del PROGRAMMA

→ INDIRIZZO dell'oggetto SUCCESSIVO IN MEMORIA.

→ Leggo dal REGISTRO o SPOSTO dal REGISTRO SORGENTE!

→ SPIAZZAMENTO RISPETTO all'IP dopo la fase di fetch!

↓

OPERANDO M