Lezione 26 C e Assembly 4

mercoledì 29 novembre 2023 14:10

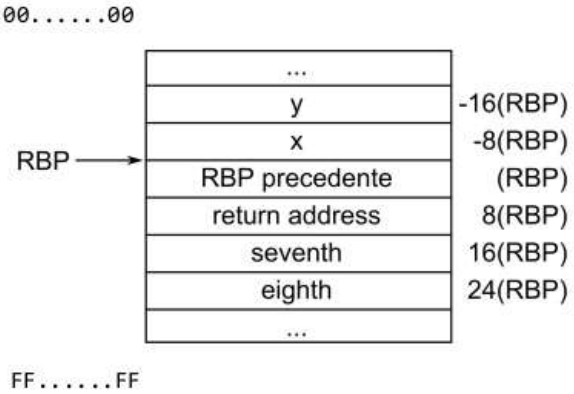
Andiamo ora a vedere come vengono passati i parametri nelle funzioni : **calling convection .** Quindi si parla di modalità di uso dei registri per passare i parametri : in generale sono "limitati" , quindi si opta per passaggio tramite stack, registri + stack e misto. Il valore di ritorno anche lui viene salvato in un registro general purpose . in dettaglio i primi 6 registri servono per passare i 6 parametri ; gli altri parametri rimanenti devono essere messi sullo stack , ma in ordine inverso rispetto a come compaiono **,** mentre per quelli di ritorno RAX e RDX . Quindi i registri si dividono in due **callee-save (salvati dal chiamato) ,** per ripristino prima del ritorno al chiamante**:** RBP,RBX, R12->R15, e **caller-save :** tutti gli altri. In dettaglio :

- I primi sei parametri (interi e indirizzi) di una subroutine vengono passati tramite registri:
  - RDI, RSI, RDX, RCX, R8, R9
- Se una subroutine accetta più di sei parametri, si utilizza lo stack per quelli aggiuntivi
- Si utilizzano due registri per il valore di ritorno:
  - RAX e RDX

- I registri sono divisi in *callee save* e *caller save*
  - callee-save: RBP, RBX, R12–R15
  - caller-save: tutti gli altri

Attenzione che il valore massimo di ritorno è 128 bit, quindi devo "spezzare" il dato usando due registri a 64 bit . Vediamo un esempio :

```
void f(int first, int second, int third, int fourth, int fifth,
       int sixth, int seventh, int eighth,) {
       int x, y;
       ...
}
```



X e Y sono variabili globali . Il chiamante li setta : i primo 6 parametri nei parametri secondo calling convection . Mentre il settimo e l'ottavo sono invertiti nello stack, in quanto ogni elemento rappresenta una distanza dalla base via via crescente . Nella cella più bassa dello stack , vi è l'indirizzo dell'istruzione successiva della CALL (fatto dalla ret) , il quale rappresenta l'indirizzo di RIP durante la fase di fetch. **Andiamo a vedere ora i tipi di dato in C**: ci sono 4 tipi di dato : **char** (quantità minima di memoria che il processore indirizza) ,**int**, **float** , **double.** Vediamo ora come definire in modo univoco le grandezze dei dati, evitando incomprensioni tra sistemi e processori : usando altri tipi di dato , ma anche cosi' non riesco a bypassare : **signed, unsigned , short e long.** Attenzione al tipo void : la funzione non restituisce nulla . In dettaglio :

Tipo	Spiegazione	Dimensione minima (bit)	Specificatore di formato
`char`	Minima unità indirizzabile della macchina che contiene l'insieme basilare di caratteri. È di tipo intero. Può essere segnato o non segnato. Contiene `CHAR_BIT` bit.	8	`%c`
`signed char`	Come un char, ma segnato. Contiene almeno l'intervallo [−127, +127].	8	`%c` (`%hhi` per il numero)
`unsigned char`	Come un char, ma non segnato. Contiene almeno l'intervallo [0, 255].	8	`%c` (`%hhu` per il numero)

`short`
`short int`
`signed short`
`signed short int`
	Tipo intero segnato breve. Contiene almeno l'intervallo [−32,767, +32,767].	16	`%hi` o `%hd`

`unsigned short` `unsigned short int`	Tipo intero non segnato breve. Contiene almeno l'intervallo [0, 65,535].	16	`%hu`

`int`
`signed`
`signed int`
	Tipo intero segnato di base. Contiene almeno l'intervallo [−32,767, +32,767]	16	`%i` o `%d`

perché convenzionalmente lo usiamo come se fosse a 32 bit?

Tipo	Spiegazione	Dimensione minima (bit)	Specificatore di formato
`unsigned` `unsigned int`	Tipo intero non segnato di base. Contiene almeno l'intervallo [0, 65,535].	16	`%u`

`long`
`long int`
`signed long`
`signed long int`
	Tipo intero segnato lungo. Contiene almeno l'intervallo [−2,147,483,647, +2,147,483,647].	32	`%li` o `%ld`

`unsigned long` `unsigned long int`	Tipo intero non segnato lungo. Contiene almeno l'intervallo [0, 4,294,967,295]	32	`%lu`

`long long`
`long long int`
`signed long long`
`signed long long int`
	Tipo intero segnato lungo lungo. Contiene almeno l'intervallo [−9,223,372,036,854,775,807, +9,223,372,036,854,775,807] (dal C99)	64	`%lli` o `%lld`

`unsigned long long` `unsigned long long int`	Tipo intero non segnato lungo lungo. Contiene almeno l'intervallo [0, +18,446,744,073,709,551,615] (dal C99)	64	`%llu`
`float`	Tipo reale a singola precisione. Le sue proprietà non sono specificate dallo standard.		`%f`
`double`	Tipo reale a doppia precisione. Le sue proprietà non sono specificate dallo standard.		`%lf` o `%f`
`long double`	Tipo reale a precisione estesa. Le sue proprietà non sono specificate dallo standard.		`%Lf` o `%LF`

Attenzione comunque ai warning : chiedo al compilatore di farmi vedere i warning : -Wall –Wextra.
Inoltre è possibile passare da un tipo di dato all'altro : si parla di **casting** . Vediamo un esempio :

```c
#include <stdio.h>

int main(int argc,char *argv[])
{
        double result=1/2*2;
        printf("result is : %lf\n",result);
        //valore in virgola mobile
        double result1 = 1. / 2 * 2;
        printf("Result1 is %lf\n", result1);
        double result2 = (double)1 / 2 * 2;
        printf("Result2 is %lf\n", result2);
        double result3 = 1 / 2 * (double)2;
        printf("Result3 is %lf\n", result3);
        return 0;
}
```

```
result is : 0.000000
Result1 is 1.000000
Result2 is 1.000000
Result3 is 0.000000
```

Inoltre le classi in C , possono avere degli identificatori :

auto: allocazione automatica (fondamentalmente inutile)

register: allocazione automatica. Suggerisce anche al compilatore di inserire l'oggetto in un registro della CPU (obsoleto)

static: linking interno: non è consentito accedere all'oggetto dall'esterno del modulo C o della funzione

extern: linking esterno: si indica al compilatore che l'oggetto è stato/sarà definito in un altro modulo C, affidando al linker il compito di rilocarlo (ridondante)

Se vogliamo avere una rappresentazione univoca (importando la libreria stdin.h) :

Tipo	Definizione
int8_t	Intero segnato a 8 bit
uint8_t	Intero non segnato a 8 bit
int16_t	Intero segnato a 16 bit
uint16_t	Intero non segnato a 16 bit
int32_t	Intero segnato a 32 bit
uint32_t	Intero non segnato a 32 bit
int64_t	Intero segnato a 64 bit
uint64_t	Intero non segnato a 64 bit

Andiamo a fare un riepilogo attraverso un esempio : file if.c

```c
#include<stdio.h>
int main(int argc,char *argv[])
{
        int i=0;
        if(argc==1)
        {
                printf("you have only one argument \n");
                printf("parameter @ position [%d] %s \n",i,argv[i]);
        }else if(argc>1)
        {
                printf("Here's your argument :\n");
                for (;i<argc;i++)
                {
                        printf("parameter @ position [%d] %s \n",i,argv[i]);
                }
                printf("\n");
        }
        return 0;
}
```

```
daniele@daniele-Aspire-E5-571G:~/Scrivania/programmi_calcolatori/esercizi_c$ gcc -Wall -Wextra if.c -o if
daniele@daniele-Aspire-E5-571G:~/Scrivania/programmi_calcolatori/esercizi_c$ ./if
you have only one argument
parameter @ position [0] ./if
```

```
daniele@daniele-Aspire-E5-571G:~/Scrivania/programmi_calcolatori/esercizi_c$ ./if first second third fourth fifth
Here's your argument :
parameter @ position [0] ./if
parameter @ position [1] first
parameter @ position [2] second
parameter @ position [3] third
parameter @ position [4] fourth
parameter @ position [5] fifth
```