Lezione 24 C e assembly 2

lunedì 20 novembre 2023 10:36

Andiamo a vedere in dettaglio : partiamo da u semplicissimo file.c che stampa a schermo un messaggio :

```
1.  #include <stdio.h>
2.
3.  /* This is a comment. */
4.  int main(int argc, char *argv[])
5.  {
6.      int times = 100;
7.
8.      // this is also a comment
9.      printf("Hello World! I welcome you %d times.\n", times);
10.     return 0;
11. }
```

Chiediamo al preprocessore di importare un altro file

Un commento multilinea

I vostri programmi partono sempre dalla funzione `main`. Il sistema operativo carica il programma e lo lancia da qui. Per funzionare, deve restituire un `int` ed accettare due parametri: un `int` per il numero di parametri e un array di stringhe `char *` per i parametri

Inizio di un blocco

Dichiarazione e assegnazione di variabile locale

Un commento a linea singola

Chiamata a funzione. È una funzione strana, con un numero arbitrario di parametri

Valore di ritorno della funzione. In questo caso è il valore di ritorno al sistema operativo

La fine di un blocco

La riga 1 chiede al pre-processore di includere la libreria nel nostro file . Due tipi di include : quello della riga 1 che va a cercare la libreria nelle cartelle di sistema , oppure quella tra virgolette : va a cercare la libreria anche nelle stesse cartelle . Da notare che i commenti possono essere multi-linea o singola linea. Per quanto riguarda il main è la funzione principale del nostro programma ed il sistema operativo gli delega l'inizio delle attività. Per quanto riguarda i parametri : il primo è un integer (contatore) che rappresenta in numero dei parametri , i quali parametri sono passati nel secondo parametro : vettore di stringhe. il "%d" rappresenta il valore di una qualche variabile (nel nostro caso è intero). Per quanto riguarda il valore di ritorno : se 0 tutto ok , altrimenti se diverso vi è errore. Andiamo a vedere ora il debugger : programma che si attacca ad altro programma e ne permette l'interruzione in modo selettivo : con questo strumento è possibile sia ispezionare lo stato del programma , ma anche di modificare i dati . In dettaglio :

Compilate aggiungendo l'opzione `-g`

Per lanciare il debugger: `gbd --args ./program [args]`

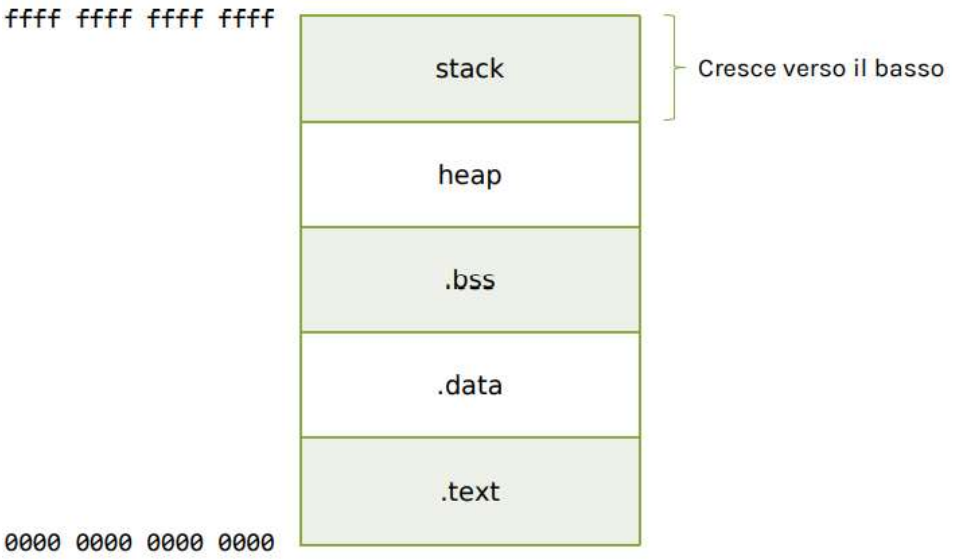
Comandi principali:

- `run [args]`: avvia il programma passando gli argomenti `[args]`.
- `break [file:]function`: imposta un breakpoint.
- `backtrace`: mostra un backtrace delle funzioni chiamate fin'ora.
- `print: expr`: mostra il valore di `expr`.
- `continue`: continua l'esecuzione del programma.
- `next`: vai alla prossima riga nel sorgente, ma non entrare nelle funzioni.
- `step`: vai alla prossima riga nel sorgente, entrando nelle funzioni.
- `step instruction`: esegui una singola istruzione assembly
- `quit`: termina l'esecuzione del debugger.

Modalità interattiva: `CTRL-x, a` (o `--tui`)

- `layout regs`: mostra una finestra con il contenuto dei registri
- `layout asm`: mostra il sorgente assembly
- `layout src`: mostra il sorgente C
- `focus [what]`: sposta il focus su una particolare finestra

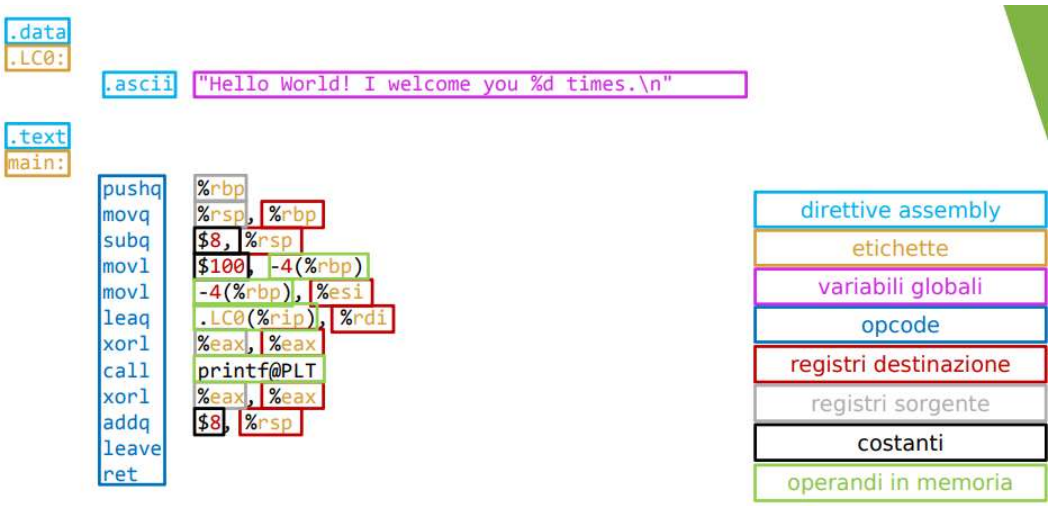
Per quanto riguarda invece al memoria :



la sezione .bss rappresenta tutte le variabili inizializzate a 0 . Nell'heap invece ci sono altre variabili .

Ed in dettaglio :

Concentriamoci ora su assembler:



Quindi in generale un programma assembler viene implementato cosi' :

```
.org [INDIRIZZO CARICAMENTO]
.data
        # Dichiarazione costanti e variabili globali

.text
main:
        # Corpo del programma
         hlt # Per arrestare l'esecuzione
```

Vediamo ora le direttive assemby :

1. **Label** : mnemonico testuale definito dal programmatore ed associato alla sequenza di istruzioni immediatamente seguenti
2. **Location counter:** viene identificato da "." : rappresenta il valore dell'indirizzo corrente; può essere usato per far saltare la "generazione di indirizzi" magando usandolo per calcolare le dimensioni delle strutture dati:
   a.
   b.
```
msg:
        .ascii "Hello, world!\\n"
len = . - msg
```
3. **.org (indirizzo), fill:** metodo alternativo per impostare il location counter, impostando i byte a fill
4. **.equ simbol, expression :** definisce una costante (non occupa memoria al momento della dichiarazione) ;
   a. Alternativa possibile : symbol=expression
   b. Possibile riusare il simbolo in più parti del codice, ma non prima della sua definizione.
5. **.byte expression:** riserva in memoria (byte) per espressione
   a.
   b.
```
var: .byte 0
array: .byte 0, 1, 2, 3, 4, 5
```
6. **.word :** analogo a .byte ma riserva memoria a 2 byte(16 bit)
7. **.longword :** analogo a .bye ma riserva memoria a 4 byte(32 bit)
8. **.quadword :** analogo a .bye ma riserva memoria a 8 byte(64 bit)
9. **.ascii "string":** riserva in memoria per un vettore di caratteri ed imposta il valore a string
10. **.fill repeat ,size:** riserva una regione di memoria composta da repeat celle di dimensione size impostata a value (default size=1 value=0).
11. **.text :** da qui in poi va nella sezione testo
12. **.data :** da qui in poi va nella sezione data
13. **.comm symbol,lenght:** dichiara un'aria di memoria con nome (symbol) di dimensione (lenght) nella sezione .bss
14. **.driver ivn:** identifica l'inizio della routine del servizio al codice ivn.