S10-L3

Traccia:
Nella lezione teorica del mattino, abbiamo visto i fondamenti del linguaggio Assembly. Dato il codice in Assembly per la CPU x86 allegato qui di seguito, identificare lo scopo di ogni istruzione, inserendo una descrizione per ogni riga di codice. Ricordate che i numeri nel formato 0xYY sono numeri esadecimali. Per convertirli in numeri decimali utilizzate pure un convertitore online, oppure la calcolatrice del vostro computer (per programmatori).

```
0x00001141 <+8>:     mov    EAX,0x20
0x00001148 <+15>:   mov    EDX,0x38
0x00001155 <+28>:   add    EAX,EDX
0x00001157 <+30>:   mov   EBP, EAX
0x0000115a <+33>:   cmp   EBP,0xa
0x0000115e <+37>:   jge     0x1176 <main+61>
0x0000116a <+49>:   mov   eax,0x0
0x0000116f <+54>:    call    0x1030 <printf@plt>
```

Analisi del codice assembly

0x00001141 <+8>: mov EAX,0x20

- Questa istruzione muove il numero esadecimale 0x20 (che equivale a 32 in decimale) nel registro EAX. Questo imposta EAX a 32.

0x00001148 <+15>: mov EDX,0x38

- Questa istruzione muove il numero esadecimale 0x38 (che equivale a 56 in decimale) nel registro EAX. Questo imposta EAX a 56.

0x00001155 <+28>: add EAX,EDX

- Questa istruzione somma il contenuto di EDX (56) al contenuto di EAX (32), il risultato (88) è poi memorizzato di nuovo in EAX.

0x00001157 <+30>: mov EBP, EAX

- Copia il valore attualmente in EAX (88, risultato dell'addizione precedente) nel registro EBP. Quindi EBP ora contiene il valore 88.

0x0000115a <+33>: cmp EBP,0xa

- Confronta il valore contenuto in EBP (88) con 0xa (che in decimale è 10). Questa istruzione è utilizzata per preparare il processore a fare un salto condizionale in base al risultato del confronto.

0x0000115e <+37>: jge 0x1176 <main+61>

- Salta all'indirizzo 0x1176 se il valore in EBP (88) è maggiore o uguale (Greater or Equal, jge) a 10. Poiché 88 è maggiore di 10, questo salto sarà eseguito.

0x0000116a <+49>: mov eax,0x0

- Se il salto non viene eseguito (in un contesto diverso da quello attuale dove il salto sarà sempre eseguito), questa istruzione resetterà il valore di EAX a 0. Questa è una preparazione tipica prima di una chiamata di sistema o di funzione per indicare uno stato o un valore di ritorno, spesso usato per indicare "nessun errore" o un'uscita pulita.

0x0000116f <+54>: call 0x1030 printf@plt

- Chiama la funzione 'printf', una funzione della libreria standard del C per stampare output formattato. L'indirizzo della funzione, 0x1030, è definito nel Procedure Linkage Table (PLT), una parte della gestione delle chiamate a funzioni esterne in un programma compilato. Il valore nel registro EAX può essere usato come parametro della funzione, a seconda di come 'printf' è utilizzato nel programma.

0 = 0000
1 = 0001
2 = 0010
3 = 0011
4 = 0100
5 = 0101
6 = 0110
7 = 0111
8 = 1000
9 = 1001
A(10) = 1010
B(11) =1011
C(12) = 1100
D(13) = 1101
E(14) = 1110
F(15) = 1111

I numeri che vedi come `+8`, `+15`, ecc., sono gli offset delle istruzioni rispetto all'inizio di una funzione o di un blocco di codice. Questi valori ti aiutano a localizzare esattamente dove si trova una certa istruzione nel codice sorgente o in un file eseguibile compilato. Ecco una spiegazione più dettagliata:

Offset di Istruzione

1. **Funzione dell'Offset**: L'offset, espressi come `+numero`, indica la distanza in byte dal punto di ingresso della funzione o del blocco di codice corrente fino all'istruzione specifica. Questo è particolarmente utile per il debug e l'analisi del codice assembly per comprendere la struttura del codice e tracciare l'esecuzione.
2. **Esempio Pratico**:
    - Supponiamo che l'indirizzo di ingresso di una funzione sia `0x00001141`. Se un'istruzione è etichettata come `0x00001141 <+8>`, significa che questa istruzione si trova 8 byte dopo l'inizio della funzione. L'indirizzo effettivo dell'istruzione sarà quindi `0x00001141 + 0x8 = 0x00001149`.