S10-L3

(Bonaldi Cristian)

Assembly x86

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>
```

Esercizio e sviluppo:

Nell’esercizio di oggi andremo ad identificare lo scopo di ogni istruzione riportata nella traccia, inserendo una descrizione per ogni riga di codice.

---

mov EAX,0x20

Convertiamo il valore esadecimale 0x20 in decimale, ottenendo il valore 32.
L’istruzione mov EAX,0x20 quindi copierà il valore 32 nel registro EAX.
In questo modo stiamo impostando EAX a 32.

---

mov EDX,0x38

Convertiamo il valore esadecimale 0x38 in decimale, ottenendo il valore 56.
L’istruzione mov EDX,0x38 quindi copierà il valore 56 nel registro EDX.
In questo modo stiamo impostando EDX a 56.

---

add EAX,EDX

L'istruzione add fa la somma tra il valore contenuto in EAX e il valore contenuto in EDX. Quindi in questo caso andremo a fare la somma tra i due valori, rispettivamente 32+56=88 e aggiorneremo il registro EAX con il valore 88.

---

mov EBP, EAX

Adesso il valore contenuto nel registro EAX andremo a copiarlo nel registro EBP, quindi il valore aggiornato di EBP sarà 88.

---

cmp EBP,0xa *EBP destinazione(88) 0xa sorgente(10)*

Ora andiamo ad eseguire un cmp tra il valore contenuto in EBP (88) e il valore esadecimale 0xa (10 in decimale). Questo significa che andremo a sottrarre il valore 10 dal valore contenuto nel registro EBP (88) e considereremo la modifica all'interno del registro EFLAGS senza modificare EBP. In questo caso ZF (Zero Flag) e CF (Carry Flag). Quindi 88-10 = 78.

cmp in questo caso prenderà in considerazione solamente le flags con valori rispettivamente:
ZF = 0 e **CF = 0**.

---

jge 0x1176 <main+61>

In questo caso, data la destinazione EBP maggiore rispetto alla sorgente 0xa dell'istruzione cmp (88 > 10) si effettua un jge alla locazione indicata, ovvero 0x1176 (4470).
jge sta per Jump if greater or equal (salta alla locazione specificata se la destinazione è maggiore o uguale rispetto alla sorgente nell'istruzione "cmp").
<main+61> ci indica semplicemente che la locazione sulla quale stiamo jumpando si trova a 61 byte dopo l'inizio della funzione main nel codice del programma.

---

mov eax,0x0

questa istruzione indica di copiare il valore 0x0 (0 in decimale) nel registro EAX. Il registro EAX conterrà quindi il valore 0.

---

call 0x1030 <printf@plt>

L’istruzione call indica la chiamata di una funzione da un indirizzo di memoria specificato nell’operando (in questo caso 0x1030).
Quindi, 0x1030 è l’indirizzo di memoria della funzione che deve essere chiamata.
printf è appunto la funzione che va a chiamare, e **@plt (Procedure Linkage Table)** stabilisce che la funzione è una libreria esterna e il linkaggio potrebbe essere gestito dinamicamente a runtime, ovvero il collegamento avviene mentre il programma è in esecuzione, non al momento della compilazione. In sostanza serve per linkare dinamicamente le chiamate a funzioni esterne.