

Esercizio 3 Settimana 10

Linguaggio Assembly

Nella lezione teorica del mattino, abbiamo visto i fondamenti del linguaggio Assembly. La traccia ci fornisce il codice in Assembly per la CPU x86 allo scopo di identificare l'azione di ogni istruzione così da fornire una descrizione per ogni riga di codice. In questo caso i numeri nel formato 0xYY sono numeri esadecimali, dove YY rappresenta la versione esadecimale del numero decimale. I registri, invece, sono il tipo di memoria ad accesso più rapido in quanto essi sono programmati all'interno della CPU, di conseguenza il percorso delle informazioni per arrivare dalla CPU ai registri è minimo, ma possono contenere informazioni solo nell'ordine dei bit.

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

La prima parte di codice identifica l'indirizzo di memoria che occuperà l'operazione assembly dopo: sarà spostata di 8 byte rispetto a quella nota.

Questa istruzione ha lo scopo di spostare tramite l'operazione mov il valore 0x20 – espresso in numeri esadecimali – nel registro EAX; il valore 0x20 corrisponde al numero decimale 32, ergo al registro EAX viene assegnato il valore 32

EAX= 32

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

Questa istruzione ha lo scopo di spostare tramite l'operazione mov il valore esadecimale 0x38 nel registro EDX; il valore 0x38 corrisponde al numero decimale 56, ergo, al registro EDX viene assegnato il valore 56

EDX= 56

0x00001155 <+28>: add EAX, EDX

Questa istruzione ha lo scopo di sommare i valori dei registri EAX ed EDX, aggiornando poi il valore del registro EAX con la somma dell'operazione. Sappiamo che il valore di EAX in numeri decimali è 32 e quello di EDX è 56, pertanto sommando i due valori otteniamo il nuovo valore 88, che viene assegnato al registro EAX

EAX= 88

0x00001157 <+30>: mov EBP, EAX

Tramite questa istruzione viene spostato tramite l'operazione mov il valore di EAX all'interno del registro EBP, che assumerà quindi il valore di 88

EBP= 88

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

In questa riga viene introdotta l'istruzione cmp, che si comporta in maniera simile all'istruzione sub, usata per sottrarre i valori di 2 registri, senza però andare a modificare i due operandi. Tuttavia questa operazione va a modificare lo ZERO FLAG (ZF) ed il CARRY FLAG (CF), che si usa per gestire eventuali riporti in un'operazione aritmetica. Questi sono degli STATUS FLAG, valori che possono avere solo valori 1 o 0 in base alle seguenti casistiche:

1. Destinazione = sorgente: ZF 1 CF 0 Se la sorgente è uguale alla destinazione, si avrà una sottrazione tra due numeri uguali es. sub 5, 5 = 0. Visto che il risultato è 0, lo ZF viene settato a 1
 2. Destinazione < sorgente: ZF 0 CF 1 Se la destinazione è minore della sorgente, si avrà una sottrazione come sub 2, 5 = -3 che verrà gestito come un numero con "prestito" (riporto), quindi CF è 1 e ZF è 0
 3. Destinazione > sorgente: ZF 0 CF 0 Se la destinazione è maggiore della sorgente, sia ZF che CF sono 0.
- In questa specifica riga di comando, l'operazione è $88 \text{ (EBP)} - 10 \text{ (0xa)} = 78$. Ne consegue che ZF e CF assumono entrambi valore 0.

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

In questa riga viene introdotta l'istruzione CONDITIONAL JUMP (salto condizionale). In questo caso l'istruzione consiste nel salto alla locazione di memoria 0x1176 nel caso in cui la destinazione abbia un valore maggiore o uguale al valore dell'istruzione cmp contenuta nella riga vista precedentemente.

Dato che $88 \geq 10$ si salta all'indirizzo di memoria 0x1176

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

In questa riga, tramite l'istruzione mov viene assegnato il valore esadecimale 0x0, che in sistema decimale equivale a 0, al registro EAX

EAX = 0

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

Nel linguaggio assembly una funzione viene chiamata con l'istruzione CALL: la funzione chiamante passa l'esecuzione del programma alla funzione chiamata per la quale verrà creato un nuovo stack.

In questo caso la funzione chiamata è PRINTF.