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.

```
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>
```

1. **mov EAX, 0X20**

mov → è l'abbreviazione di "move" (sposta) e indica all'assemblatore che vogliamo copiare un valore in un'altra posizione.

EAX → è uno dei registri generali della CPU x86, che viene comunemente utilizzato per varie operazioni aritmetiche e di manipolazione dei dati.

0x20 → convertiamo il numero esadecimale 0x20 in decimale: 32, ovvero il valore costante che vogliamo copiare nel registro EAX.

Quindi, l'istruzione mov EAX,32 sposta il valore 32 nel registro EAX.

2. **mov EDX, 0X38**

mov → copia

EDX → è uno dei registri generali della CPU x86, che viene spesso utilizzato per contenere valori temporanei o risultati di operazioni.

0x38 → convertito in decimale è 56, ovvero è il valore costante che vogliamo copiare nel registro EDX.

Quindi, l'istruzione mov EDX, 56 sposta il valore 56 nel registro EDX. In altre parole, EDX ora contiene il valore 56 che può essere utilizzato nelle successive operazioni del programma Assembly.

3. **add EAX, EDX**

add —> è un'istruzione che indica di eseguire un'operazione di aggiunta. Quindi, l'istruzione add EAX, EDX somma il contenuto del registro EDX al registro EAX e memorizza il risultato in EAX. Quindi, il valore in EDX viene aggiunto al valore in EAX, e il risultato viene memorizzato nuovamente in EAX.

4. **mov EBP, EAX**

E' un'istruzione Assembly che copia il contenuto del registro EAX nel registro EBP. EBP → è un registro della CPU x86 spesso utilizzato come base pointer per l'accesso alle variabili locali e ai parametri delle funzioni all'interno del frame dello stack.

Quindi, l'istruzione mov EBP, EAX copia il valore attualmente contenuto nel registro EAX nel registro EBP. Ciò può essere utile per salvare temporaneamente un valore per un utilizzo successivo all'interno del programma.

5. **cmp EBP, 0xa**

E' un'istruzione Assembly che confronta il valore nel registro EBP con il valore 0xa:

cmp → è l'abbreviazione di "compare" (confronta) e viene utilizzata per confrontare due valori.

EBP → è un registro della CPU x86 che contiene generalmente un puntatore alla base del frame dello stack.

0xa → è un valore costante esadecimale (10 in decimale) che vogliamo confrontare con il valore in EBP.

L'istruzione cmp EBP, 0xa: confronta il valore nel registro EBP con il valore costante 0xa. Durante il confronto, non viene effettuata alcuna modifica ai registri; piuttosto, vengono impostati i flag della CPU in base al risultato del confronto.

6. **jge 0x1176 <main+61>**

E’ un'istruzione di salto condizionato, i quali utilizzano il contenuto dei flags per determinare se “saltare” o meno ad una data locazione che viene specificata come operando dell’istruzione jump.

jge → è l'abbreviazione di "jump if greater than or equal" (salto se maggiore o uguale) e viene utilizzata per saltare a un'etichetta specifica se il confronto precedente ha risultato maggiore o uguale.

0x1176 <main+61> → è l'indirizzo di destinazione del salto. Indica l'indirizzo della prossima istruzione da eseguire nel codice. In questo caso, main+61 suggerisce che il salto porterà l'esecuzione alla posizione 61 byte dopo l'inizio della funzione main. L'indirizzo esadecimale 0x1176 corrisponde a questo punto nel codice.

Quindi, l'istruzione jge 0x1176 <main+61> salterà all'etichetta main+61 solo se il confronto precedente ha risultato maggiore o uguale. Altrimenti, l'esecuzione proseguirà con l'istruzione successiva.

7. **mov eax, 0x0**

E’ un'istruzione che assegna il valore zero al registro EAX:

0x0 → è il valore costante esadecimale che vogliamo copiare nel registro EAX. In questo caso, 0x0 rappresenta il valore zero in esadecimale, che equivale a 0 in decimale.

Quindi, l'istruzione mov eax, 0x0 imposta il registro EAX al valore zero.

8. **call 0x1030 <printf@plt>**

call → è un'istruzione di controllo del flusso che viene utilizzata per chiamare una funzione o un sotto-programma.

0x1030 <printf@plt> → è l'indirizzo di destinazione della chiamata. Indica l'indirizzo della funzione printf presente nella Procedure Linkage Table (PLT) del programma. Questo indirizzo è rappresentato in formato esadecimale.

Quindi, l'istruzione call 0x1030 <printf@plt> esegue una chiamata alla funzione printf. Questa istruzione mette sullo stack l'indirizzo di ritorno, in modo che il controllo venga trasferito alla funzione printf. Dopo aver eseguito la funzione printf, il controllo tornerà all'istruzione successiva all'istruzione di chiamata.