

S10L3

Breve guida AL LINGUAGGIO ASSEMBLY (emulatore EMU8086)



Preparato da
SARAH ORTIZ

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>
```

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

con **MOV**, carica il valore esadecimale (0x20) nel registro EAX → decimale 32

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

con **MOV**, carica il valore 0x38 nel registro EDX → decimale 56

0x00001155 <+28>: add EAX,EDX

con **ADD**, somma il valore in EAX con EDX e poi memorizzarlo nel primo → $EAX = 32 + 56$

0x00001157 <+30>: mov EBP,EAX

con **MOV**, carica ciò che c'è in EAX in EBP → il risultato della somma (88) si trova in EBP

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

con **cmp**, confronta il valore 0xa con il valore di EBP → confronto tra 10 e 88

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

con **jge**, segue un'istruzione **cmp** eseguendo un salto condizionale all'indirizzo 0x1176 se il valore di EBP è ≥ a 10

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

con **MOV**, carica il valore 0x0 nel registro di EAX → 0 verrà copiato/caricato in EAX

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

con **CALL**, chiama un sottoprogramma di printf tramite PLT, e 0x1030 è l'indirizzo della funzione printf

0x00001141 <+8>:

0x00001148 <+15>:

0x00001155 <+28>:

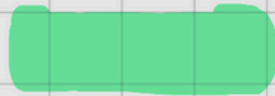
0x00001157 <+30>:

0x0000115a <+33>:

0x0000115e <+37>:

0x0000116a <+49>:

0x0000116f <+54>:

 = indirizzo di memoria

 = offset (byte)

Sono indirizzi e offset che danno informazioni sul punto di memoria, in cui
ogni istruzione si trova nel programma.

*distance tra un punto di riferimento e una particolare
posizione dentro il codice*

0x[indirizzo]	<offset>
↓	↓
00001141	+8 byte
indirizzo di memoria	