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>
```

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

In questa sezione di codice Assembly tramite l' istruzione "mov" va ad assegnare al registro EAX, registri di CPU ben definiti, il valore esadecimale 0x20 (32), il che vuol dire che il registro EAX ora avrà valore 32

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

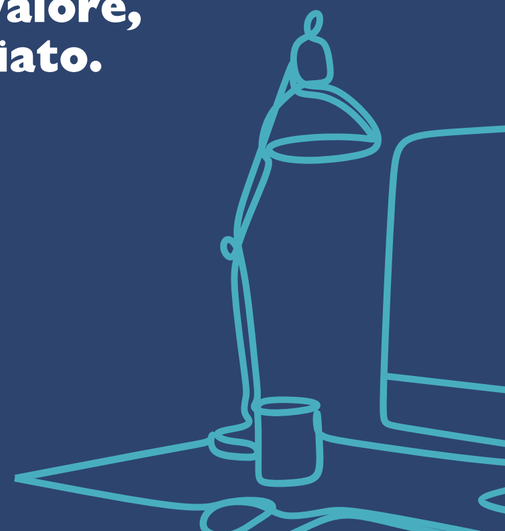
In questa sezione di codice accade la stessa cosa vista nella precedente, solo che in questo caso il registro EDX assumerà valore 0x38 (56)

0x00001155 <+28>: add EAX,EDX

In questa sezione di codice vado ad aggiungere al registro EAX (Destinazione) il valore del registro EDX (sorgente), in questo caso

32+56=88

a seguito dell’operazione il registro Destinazione assumerà il nuovo valore, mentre il sorgente rimarrà invariato.

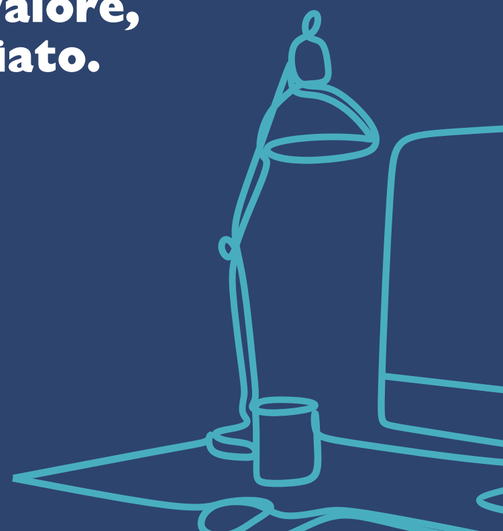


0x00001155 <+28>: add EAX,EDX

In questa sezione di codice vado ad aggiungere al registro EAX (Destinazione) il valore del registro EDX (sorgente), in questo caso

32+56=88

a seguito dell’operazione il registro Destinazione assumerà il nuovo valore, mentre il sorgente rimarrà invariato.



0x00001157 <+30>: mov EBP, EAX

In questa sezione di codice assegno il nuovo valore del registro EAX al registro EBP, a seguito di ciò sia EAX che EBP avranno valore 88

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

In questa sezione di codice l’istruzione cmp è simile asub, ma a differenza di quest’ultima non andrà a modificare gli operandi, bensì andrà a modificare i flag ZF e CF, in questo caso andremo a fare:

88 - 10 = 78

in questo caso avendo una destinazione maggiore della sorgente, le flag saranno 0 e 0

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

In questa sezione di codice andremo a utilizzare un salto condizionale, ovvero a seguito di una condizione rispettata (con jge se la destinazione è maggiore della sorgente presente in cmp) si passerà direttamente all'indirizzo di memoria indicato dall'istruzione, in questo caso " 0x1176 <main+61>"

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

In questa sezione di codice assegno il valore 0x0 (0) al registro EAX, precedentemente 88.

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

In questa sezione di codice richiamo la funzione “printf” contenuta all’interno dell’indirizzo di memoria “0x1030”