






CONSEGNA S10/L3

DI GIUSEPPE LUPOI





INDICE

- 3. TRACCIA
 - 4. IL LINGUAGGIO ASSEMBLY
 - 5. RIEPILOGO DEL CODICE
 - 6. CONVERSIONE DEI VALORI
 - 7. SPIEGAZIONE 1° PARTE
 - 8. SPIEGAZIONE 2° PARTE
- 
- 
- 

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>
```

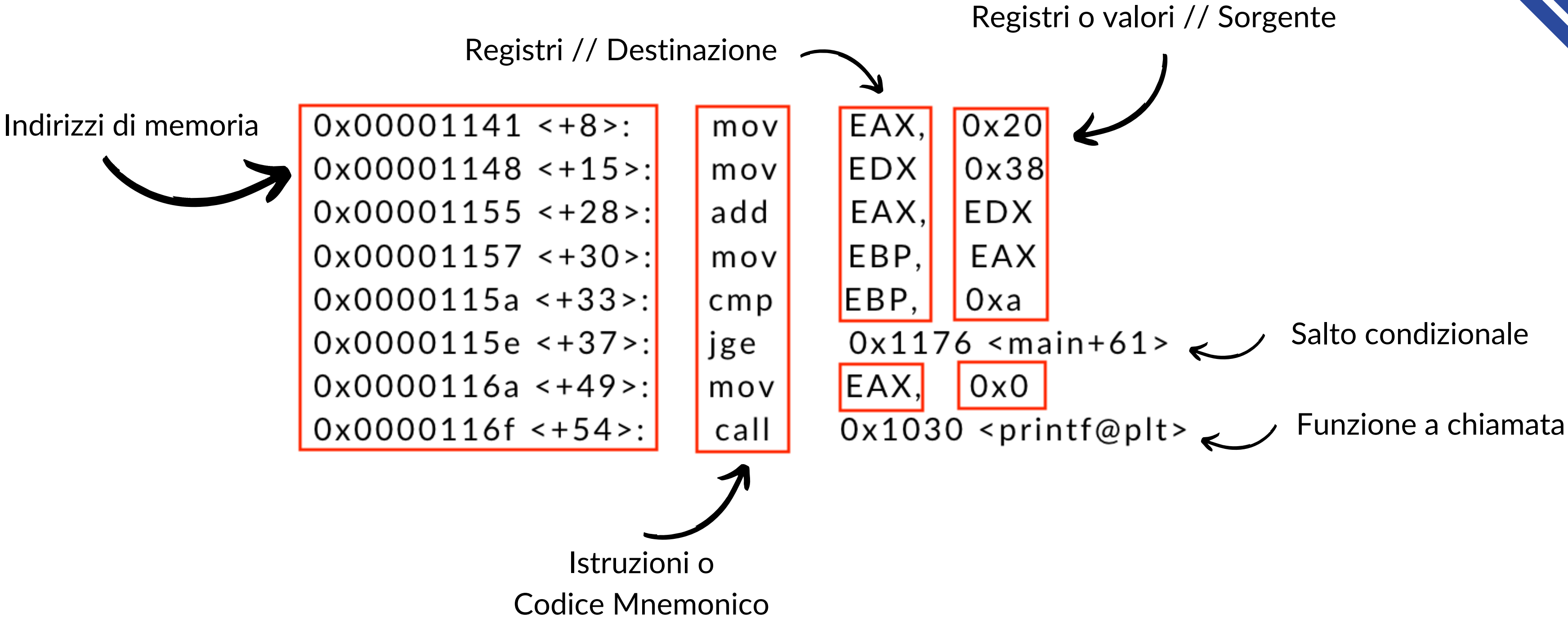
COS'È IL LINGUAGGIO ASSEMBLY?

Il linguaggio assembly è un linguaggio di basso livello che fornisce un'interfaccia tra il linguaggio macchina e il linguaggio di programmazione di più alto livello. L'assembly è specifico per l'architettura del processore su cui viene eseguito, il che significa che ci sono diverse varianti di linguaggio assembly per diversi tipi di CPU.

Ecco alcuni concetti chiave nel linguaggio assembly:

- 1. Istruzioni di Base:** Le istruzioni assembly sono direttamente correlate alle istruzioni eseguite dalla CPU. Queste istruzioni eseguono operazioni fondamentali come il trasferimento di dati tra registri, l'aritmetica binaria, il controllo del flusso (salti condizionati o incondizionati) e altro.
 - 2. Registri:** I registri sono piccole aree di memoria all'interno della CPU utilizzate per archiviare dati temporanei o informazioni di controllo. Nel linguaggio assembly, spesso si fa riferimento ai registri utilizzando abbreviazioni come EAX, EBX, ECX, EDX, etc.
 - 3. Direttive e Istruzioni di Assemblaggio:** Le direttive sono istruzioni che non vengono eseguite dalla CPU ma forniscono informazioni al compilatore. Ad esempio, possono essere usate per riservare spazio di memoria per variabili. Le istruzioni di assemblaggio, d'altra parte, sono le istruzioni effettive eseguite dal processore.
 - 4. Operandi:** Gli operandi sono i dati su cui le istruzioni di assemblaggio agiscono. Possono essere costanti, indirizzi di memoria, registri o espressioni complesse.
- Etichette e Salti: Le etichette sono nomi dati a posizioni specifiche nel codice sorgente. Possono essere utilizzate per indicare punti di destinazione per salti condizionati o incondizionati.

Prima di procedere con lo svolgimento dell'esercizio rivedo in blocco il codice Assembly a noi fornito per fare chiarezza.



Per prima cosa per semplificarci la situazione procediamo con la conversione dei valori da numeri esadecimali a decimali.

Quindi:

```
0X00001141 <+8>:  MOV    EAX,0X20 ← 32
0X00001148 <+15>: MOV    EDX,0X38 ← 56
0X00001155 <+28>:  ADD    EAX,EDX
0X00001157 <+30>:  MOV    EBP, EAX
0X0000115A <+33>:  CMP    EBP,0XA ← 10
0X0000115E <+37>:  JGE    0X1176 ← 4470 <MAIN+61>
0X0000116A <+49>:  MOV    EAX,0X0 ← 0
0X0000116F <+54>:  CALL   0X1030 ← 4144 <PRINTF@PLT>
```

Dopo aver fatto ciò possiamo procedere con la spiegazione del codice riga per riga.

Avremo quindi:

0x00001141 <+8>: mov EAX, 32 ← istruzione “move” quindi il valore 32 verrà copiato nel registro EAX

0x00001148 <+15>: mov EDX, 56 ← istruzione “move” quindi il valore 56 verrà copiato nel registro EDX

0x00001155 <+28>: add EAX, EDX ← istruzione “add” quindi il valore di EDX verrà sommato a quello di EAX, ed il valore di EAX verrà quindi aggiornato

0x00001157 <+30>: mov EBP, EAX ← istruzione “move” quindi il valore EAX verrà copiato nel registro EBP

Avremo quindi:

0x0000115a <+33>: cmp EBP, 10 ← istruzione “cmp”, simile ad una sottrazione, quindi 10 verrà sottratto al valore di EBP per poi modificare da 0 o 1 i valori dei flag ZERO o CARRY.

0x0000115e <+37>: jge 4470 <main+61> ← salto condizionale “jge” che salterà alla locazione specificata solo se la destinazione è maggiore o uguale della sorgente nell’istruzione “cmp” precedente

0x0000116a <+49>: mov EAX, 0 ← istruzione “move” quindi il valore 0 verrà copiato nel registro EAX

0x0000116f <+54>: call 4144 <printf@plt> ← funzione “call” che appunto richiamerà dentro il codice la funzione “printf”