ASSEMBLY BASICS

S10 - L3

```
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>
```

Questa é la prima riga del codice che verra eseguita."mov" è una delle istruzioni che Assembly può eseguire. In questo caso viene utilizzato per indicare che al registro EAX deve essere assegnato il numero esadecimale 0x20 (32 in decimale)

Questa sezione indica semplicemente l'indirizzo di memoria nella tabella relazionale in cui si trova ciascuna riga di questo codice Assembly x86.

```
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>
```

Questa riga svolge la stessa funzione dell'ultima, ma con valori diversi. Assegna il valore esadecimale 0x38 (56 in decimale) al registro EDX, a traverso di l'istruzione mov

```
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>
```

Questa riga, tramite l'istruzione ADD, somma il valore interno al registro EDX al valore interno al registro EAX, cioè 32 + 56, con risultato decimale di 88. Dico che EDX viene sommato a EAX perché il risultato di questa addizione è memorizzato all'interno di EAX stesso.

```
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>
```

Questa riga, con l'istruzione **mov**, come visto in precedenza, sposta un valore in un registro, in questo caso sposta **EAX** (con il valore attuale di &&) nel registro **EBP**.

```
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>
```

Nel caso in cui il salto non sia stato effettuato il codice continuerà ad essere interpretato normalmente. La riga successiva che verrebbe interpretata sarebbe quindi quella che indica lo spostamento del valore di 0x0 (0 in decimale) nel registro eax.

```
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>
```

Infine, l'ultima riga di questo codice chiama la funzione printf all'indirizzo 0x1030, presumibilmente per fornire un messaggio in output

```
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>
```

In conclusione queste righe di codice sembrano far parte di una funzione o di un programma più ampio. Il codice assembly esegue operazioni utilizzando i registri ed esegue ramificazioni condizionali in base al risultato del confronto.