**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 🡪** Carica il valore esadecimale 0x20 (32 in decimale) nel registro EAX;

**0x00001148 <+15>: mov EDX,0x38 🡪** Carica il valore esadecimale 0x38 (56 in decimale) nel registro EDX;

**0x00001155 <+28>: add EAX,EDX 🡪** Aggiunge il contenuto del registro EDX al registro EAX e memorizza il risultato in EAX;

**0x00001157 <+30>: mov EBP, EAX 🡪** Copia il valore contenuto nel registro EAX nel registro EBP;

**0x0000115a <+33>: cmp EBP,0xa 🡪** Compara il valore contenuto nel registro EBP con il valore esadecimale 0xa (10 in decimale);

**0x0000115e <+37>: jge 0x1176 <main+61> 🡪** Salta all'indirizzo 0x1176 se il risultato della comparazione precedente indica che EBP è maggiore o uguale a 10;

**0x0000116a <+49>: mov eax,0x0 🡪** Carica il valore esadecimale 0x0 (0 in decimale) nel registro EAX;

**0x0000116f <+54>: call 0x1030 <printf@plt> 🡪** Chiama la funzione printf situata all'indirizzo 0x1030 nel programma (nell'area chiamata "PLT", Procedure Linkage Table). Presumibilmente, il risultato di questa operazione sarà stampato sulla console.

Per comprendere al meglio l’esercizio appena svolto è bene fare alcune considerazioni.

Prima di tutto cos’è il linguaggio *Assembly.* Il linguaggio Assembly è un linguaggio di programmazione di basso livello che fornisce un'interfaccia simbolica per le istruzioni di macchina di un processore. È molto vicino al linguaggio macchina, il linguaggio binario comprensibile al processore stesso.

Ad esempio ipotizziamo di avere un programma scritto in C. Il *compilatore* traduce le istruzioni scritte nel **linguaggio C in linguaggio macchina** e le salva in un file eseguibile, che viene letto dalla CPU che conosce solo il linguaggio macchina. Il *disassembler* esegue il processo inverso, riportando però le istruzioni in linguaggio Assembly, leggibili dall’uomo.

La nostra attenzione si è focalizzata sull’architettura di processori **x86 a 32 bit** chepossono quindi eseguire istruzioni di dimensioni fino a 32 bit. Esso si avvale di diversi registri, ovvero una piccola area di memoria all'interno della CPU utilizzata per memorizzare dati temporanei, istruzioni di macchina in esecuzione o informazioni di stato del processore, ma i principali sono:

* **EAX**: Registro accumulatore esteso, utilizzato per molte operazioni aritmetiche e logiche.
* **EBX**: Registro base esteso, spesso utilizzato come puntatore a dati o per memorizzare valori temporanei.
* **ECX**: Registro contatore esteso, utilizzato spesso come contatore in cicli o loop.
* **EDX**: Registro dati esteso, utilizzato per estendere la capacità del registro EAX per alcune operazioni.
* **ESI**: Registro sorgente indice, utilizzato in operazioni di copia dei dati.
* **EDI**: Registro destinazione indice, utilizzato in operazioni di copia dei dati.
* **EBP**: Registro base del puntatore esteso, spesso utilizzato come base per l'accesso alle variabili locali o ai parametri delle funzioni.
* **ESP**: Registro stack pointer esteso, utilizzato per tenere traccia dell'indirizzo corrente nello stack.
* **EIP**: Registro instruction pointer esteso, contiene l'indirizzo di memoria dell'istruzione successiva da eseguire.
* **EFLAGS**: Registro flag esteso, contiene i flag di stato del processore che indicano il risultato di operazioni precedenti.

A seconda dell’architettura del processore, il linguaggio Assembly utilizza delle istruzioni, simboli comprensibili che eseguono operazioni di varia natura.

Le più utilizzate sono:

1. TRASFERIMENTO DATI 🡪 **mov** (trasferimento diretto), **push** (inserimento nello stack), **pop** (rimozione dallo stack);
2. ARITMETICHE E LOGICHE 🡪 **add**, **sub**, **mul**, **div**, **and**, **or**, **not**, **xor;**
3. CONTROLLO DEL FLUSSO 🡪 **jmp**, **je** (jump se uguale), **jne** (jump se non uguale), **jg** (jump se maggiore), **jl** (jump se minore), **call** (chiamata di una funzione), **ret** (ritorno da una funzione);
4. MANIPOLAZIONE DELLO STACK 🡪 **push** (inserimento nello stack) e **pop** (rimozione dallo stack), **call** (chiamata di una funzione), **ret** (ritorno da una funzione);
5. INPUT/OUTPUT 🡪 **in** (input da un dispositivo), **out** (output a un dispositivo), e **int** (interruzioni software per richiedere servizi di sistema).

In linguaggio assembly, vengono utilizzati diversi formati anche per rappresentare numeri:

1. **Decimale**: come **123**, **456**, etc. Questo è il formato numerico più comune e viene utilizzato per esprimere valori in base 10;
2. **Esadecimale**: preceduti dal prefisso **0x**, come **0xAB**, **0x10**, etc. Questo è un formato comune per esprimere costanti numeriche o indirizzi di memoria in base 16;
3. **Binario**: preceduti dal prefisso **0b**, come **0b1010**, **0b1100**, etc. Questo formato è utile per esprimere valori in base 2.