Esercizio S10/L3

Traccia:
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.

Il linguaggio **Assembly** serve a leggere le istruzioni eseguite dalla CPU in formato leggibile all'uomo ed è univoco per una data architettura di un PC, ma cambia da architettura ad architettura.
Il linguaggio Assembly mette a disposizione le istruzioni per spostare dati dalla memoria ai registri, istruzioni per calcoli aritmetici, operazioni logiche sui bit e istruzioni per i salti condizionali.
Le architetture della CPU sono **x86 (32bit) o 64 (64bit)**.
L'architettura x86 mette a disposizione registri a 32 bit.

0x00001141 <+8>: **mov EAX,0x20**:
Sposta il valore **32** dentro il registro **EAX**. Il prefisso 0x indica il linguaggio di programmazione del codice

0x00001148 <+15>: **mov EDX,0x38**
Sposta il valore **56** nel registro dati per accedere alle porte di I/O o funzioni aritmetiche

0x00001155 <+28>: **add EAX,EDX**
Somma il valore EDX con EAX salvando/aggiornando EAX con il valore **88**

0x00001157 <+30>: **mov EBP, EAX**
Sposta il valore **88** sul registro puntatore base

0x0000115a <+33>: **cmp EBP,0xa**
Compara il valore **10** con il valore del puntatore base **88** indicando che i flag **ZF** e **CF** saranno **0**

0x0000115e <+37>: **jge 0x1176 <main+61>**
Salta alla locazione **4470**, perché la destinazione è maggiore della sorgente nell'istruzione **cmp**

0x0000116a <+49>: **mov EAX,0x0**
Spoata il valore **0** dentro il registro **EAX**

0x0000116f <+54>: **call 0x1030 <printf@plt>**
Chiama la funzione **4144** dalla **Procedure Linkage Table** (**plt**), che contiene gli indirizzi assoluti delle funzioni esterne al file

Legenda:
0x: indica che il codice è scritto in linguaggio C o Java;
EAX: accumulatore;
EDX: Registro dati, usato per accedere alle porte di I/O e per le funzioni aritmetiche;
EBP: Registro puntatore di base;
mov: Sposta una variabile o un dato da una locazione ad un'altra, utilizzata per leggere e scrivere in memoria;
add: somma 2 valori e salva/aggiorna il valore destinatario;
cmp: simile a «sub» e non modifica gli operandi, ma modifica i flag Zero Flag (ZF) e Carry Flag (CF);
jge: Salta alla locazione specificata se la destinazione è maggiore o uguale della sorgente nell'istruzione «cmp»;
call: la funzione chiamante passa l'esecuzione alla funzione chiamata per generare un nuovo stack.
EFLAGS: registro utilizzato per prendere decisioni sulla base del valore di un determinato flag
ZF (Zero Flag): Indica se il risultato di un'operazione matematica o logica è zero
CF (Carry Flag): Indica se il risultato di un'operazione produce una risposta non contenibile nei bit usati per il calcolo