CS0424IT — ESERCITAZIONE S10L3
LINGUAGGIO ASSEMBLY

Simone La Porta

GitHub

---

31 luglio 2024

INDICE

1 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>
```

In questo report, si analizza un frammento di codice scritto in linguaggio Assembly per la CPU x86. Questo esercizio ha l'obiettivo di comprendere il funzionamento delle istruzioni Assembly, i loro effetti sui registri e sul flusso di controllo del programma.

2 DESCRIZIONE DEL LINGUAGGIO ASSEMBLY

Il linguaggio Assembly è un linguaggio di basso livello utilizzato per programmare direttamente la CPU. Ogni istruzione Assembly corrisponde a una singola operazione eseguita dalla CPU, come il caricamento di valori nei registri, operazioni aritmetiche, confronti, salti condizionali, ecc. Il linguaggio Assembly è specifico per l'architettura della CPU per cui è scritto, e nel nostro caso ci focalizziamo sull'architettura x86.

2.1 *Basi dei comandi Assembly*

Per comprendere il codice Assembly, è importante conoscere alcune istruzioni di base:

2.1.1 MOV

L'istruzione `MOV` copia un valore da una sorgente a una destinazione. La sintassi è:

```
MOV destinazione, sorgente
```

La sorgente può essere un valore immediato, un registro o un indirizzo di memoria, mentre la destinazione è solitamente un registro o un indirizzo di memoria.

2.1.2 ADD

L'istruzione `ADD` somma il valore della sorgente al valore della destinazione e memorizza il risultato nella destinazione. La sintassi è:

```
ADD destinazione, sorgente
```

2.1.3 CMP

L'istruzione `CMP` confronta due operandi sottraendo la sorgente dalla destinazione. Il risultato della sottrazione viene utilizzato per impostare i flag della CPU, che determinano il flusso di esecuzione successivo. La sintassi è:

```
CMP operando1, operando2
```

2.1.4 JGE

L'istruzione `JGE` (Jump if Greater or Equal) trasferisce il controllo del programma a un'altra istruzione se la condizione specificata dai flag della CPU è vera. In particolare, salta se il risultato dell'ultima istruzione di confronto è maggiore o uguale a zero. La sintassi è:

```
JGE etichetta
```

2.1.5 CALL

L'istruzione `CALL` trasferisce il controllo del programma a una subroutine (funzione) specificata. La sintassi è:

```
CALL etichetta
```

2.2 *Struttura dei comandi Assembly*

Ogni riga di codice Assembly può essere scomposta in diverse parti:

- **Indirizzo:** `0x00001141`

  Questo è l'indirizzo di memoria dove l'istruzione è memorizzata. In un programma compilato, ogni istruzione ha un indirizzo unico.

- **Offset:** <+8>

  Questo rappresenta l'offset rispetto all'inizio della funzione o del blocco di codice corrente. L'offset indica quanto dista l'istruzione corrente dall'inizio della funzione.

- **Istruzione:** `mov EAX,0x20`

  Questa è l'istruzione Assembly vera e propria, composta dall'operazione (`mov`) e dagli operandi (`EAX,0x20`).

3 SVOLGIMENTO

Descrizione delle operazioni:

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

  Indirizzo: `0x00001141`

  Offset: `<+8>`

  Istruzione: `mov EAX,0x20`

  L'istruzione `mov` carica un valore immediato nel registro specificato. In questo caso, il valore esadecimale `0x20` (che equivale a 32 in decimale) viene caricato nel registro `EAX`. Il registro `EAX` è uno dei registri generali della CPU x86 e viene spesso utilizzato per operazioni aritmetiche e logiche.

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

  Indirizzo: `0x00001148`

  Offset: `<+15>`

  Istruzione: `mov EDX,0x38`

  Simile all'istruzione precedente, questa `mov` carica il valore esadecimale `0x38` (56 in decimale) nel registro `EDX`. Il registro `EDX` è un altro registro generale utilizzato per varie operazioni.

- `0x00001155 <+28>: add EAX,EDX`

  Indirizzo: `0x00001155`

  Offset: `<+28>`

  Istruzione: `add EAX,EDX`

  L'istruzione `add` somma il valore del registro `EDX` al valore del registro `EAX` e memorizza il risultato nel registro `EAX`. Dopo questa operazione, `EAX` conterrà il valore 88 (32 + 56).

- `0x00001157 <+30>: mov EBP,EAX`

  Indirizzo: `0x00001157`

  Offset: `<+30>`

  Istruzione: `mov EBP,EAX`

  Questa istruzione `mov` copia il valore del registro `EAX` nel registro `EBP`. Il registro `EBP` è spesso utilizzato come puntatore di base per il frame dello stack, ma qui viene utilizzato per conservare temporaneamente il risultato della somma.

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

  Indirizzo: `0x0000115a`

  Offset: `<+33>`

  Istruzione: `cmp EBP,0xa`

  L'istruzione `cmp` confronta il valore del registro `EBP` con il valore immediato `0xa` (10 in decimale). Questa operazione sottrae implicitamente `0xa` dal valore di `EBP` e aggiorna i flag della CPU in base al risultato, senza modificare i registri coinvolti. I flag della CPU (come lo Zero Flag, il Carry Flag e l'Overflow Flag) determinano il flusso di controllo successivo del programma.

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

  Indirizzo: `0x0000115e`

  Offset: `<+37>`

  Istruzione: `jge 0x1176 <main+61>`

  L'istruzione `jge` (jump if greater or equal) verifica i flag della CPU impostati dall'istruzione `cmp` precedente. Se il valore in `EBP` è maggiore o uguale a 10, il controllo del programma salta all'indirizzo `0x1176`. Questo significa che il flusso del programma continuerà dall'istruzione situata all'indirizzo `0x1176`, saltando tutte le istruzioni intermedie.

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

  Indirizzo: `0x0000116a`

  Offset: `<+49>`

  Istruzione: `mov eax,0x0`

  Se il salto condizionale `jge` non è stato eseguito, questa istruzione `mov` carica il valore immediato `0x0` (0 in decimale) nel registro `EAX`. Questo può essere usato per indicare che una certa condizione non è stata soddisfatta.

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

  Indirizzo: `0x0000116f`

  Offset: `<+54>`

  Istruzione: `call 0x1030 <printf@plt>`

  L'istruzione `call` trasferisce il controllo alla funzione specificata all'indirizzo `0x1030`, che in questo caso è la funzione `printf`. La chiamata a `printf` è probabilmente utilizzata per stampare il risultato memorizzato in `EAX`. La funzione `printf` è una routine di libreria standard in C che stampa un messaggio formattato sullo standard output (solitamente lo schermo).

4 CONCLUSIONE

In questo esercizio si è analizzato un frammento di codice Assembly per la CPU x86, comprendendo le operazioni eseguite da ciascuna istruzione e il loro effetto sui registri della CPU. Le istruzioni analizzate includono:

- `mov EAX,0x20` e `mov EDX,0x38`: caricano valori immediati nei registri `EAX` e `EDX`, rispettivamente.
- `add EAX,EDX`: somma i valori dei registri `EAX` e `EDX` e memorizza il risultato in `EAX`.
- `mov EBP,EAX`: copia il valore di `EAX` nel registro `EBP`.
- `cmp EBP,0xa`: confronta il valore di `EBP` con 10, aggiornando i flag della CPU.
- `jge 0x1176`: salta a un'altra istruzione se `EBP` è maggiore o uguale a 10.
- `mov eax,0x0`: imposta `EAX` a 0 se il salto condizionale non è stato eseguito.
- `call 0x1030 <printf@plt>`: chiama la funzione `printf` per stampare il risultato.

Queste istruzioni illustrano come i registri della CPU vengono utilizzati per eseguire operazioni aritmetiche, confronti e controllare il flusso del programma. La comprensione del linguaggio Assembly è cruciale per chiunque lavori a basso livello con l'hardware, come nei campi della sicurezza informatica e dell'analisi del malware, poiché consente una visione dettagliata di come i programmi interagiscono direttamente con la CPU.