Report sull'esercitazione in Assembly

Introduzione

L'Assembly è un linguaggio di basso livello che permette una comunicazione diretta con il processore, fornendo un controllo preciso delle operazioni a livello hardware. In questa esercitazione, mi concentrerò sull'analisi di un segmento di codice Assembly per la CPU x86, con l'obiettivo di identificare e spiegare in dettaglio lo scopo di ciascuna istruzione.

Codice Assembly Analizzato

Il codice Assembly fornito è il seguente:

```
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>
```

Descrizione delle Istruzioni

Per analizzare ogni istruzione, ho suddiviso il codice in una tabella dettagliata che spiega il significato e lo scopo di ciascuna operazione.

ISTRUZIONE	DESCRIZIONE
`0x00001141 <+8>: mov EAX,0x20`	Copia il valore intero decimale 32 nel registro `EAX`.
`0x00001148 <+15>: mov EDX,0x38`	Copia il valore intero decimale 56 nel registro `EDX`.
`0x00001155 <+28>: add EAX,EDX`	Somma il registro `EDX` ad `EAX`, quindi somma 56 a 32 ed aggiorna il registro `EAX` con la somma, risultando in `EAX = 88`.
`0x00001157 <+30>: mov EBP,EAX`	Copia il contenuto del registro `EAX` (88) nel registro `EBP`.
`0x0000115a <+33>: cmp EBP,0xa`	Confronta (`CMP` = compare) l'uguaglianza tra il valore `0xa` (10 in decimale) con il valore contenuto in `EBP` (88). Imposta i flag ZF e CF di conseguenza.
`0x0000115e <+37>: jge 0x1176 <main+61>`	Esegue un salto condizionale se la destinazione di `cmp` è maggiore o uguale al valore di controllo (10). Poiché 88 è maggiore di 10, il salto viene effettuato.
`0x0000116a <+49>: mov eax,0x0`	Copia il valore di `EAX` con il valore `0` (0 in decimale), quindi `EAX = 0`.
`0x0000116f <+54>: call 0x1030 <printf@plt>`	Chiamata di funzione. In questo caso, la funzione `printf` viene chiamata.

Analisi Dettagliata delle Istruzioni

1. **0x00001141 <+8>: mov EAX,0x20**
   - **Descrizione:** Questa istruzione sposta il valore esadecimale `0x20` (32 in decimale) nel registro `EAX`.
   - **Scopo:** Inizializzare il registro `EAX` con il valore 32 è un passo comune per preparare un registro con un valore iniziale fisso, che può essere usato in successive operazioni aritmetiche o logiche.
2. **0x00001148 <+15>: mov EDX,0x38**
   - **Descrizione:** Questa istruzione sposta il valore esadecimale `0x38` (56 in decimale) nel registro `EDX`.
   - **Scopo:** Similmente all'istruzione precedente, questo passo inizializza il registro `EDX` con il valore 56, preparandolo per le operazioni successive.
3. **0x00001155 <+28>: add EAX,EDX**
   - **Descrizione:** Questa istruzione somma il contenuto del registro `EDX` (56) al contenuto del registro `EAX` (32), memorizzando il risultato in `EAX`. Dopo l'esecuzione, `EAX` conterrà il valore 88.
   - **Scopo:** Calcolare la somma dei due valori precedentemente caricati nei registri `EAX` e `EDX`. Questa operazione è fondamentale per preparare `EAX` per l'uso successivo.
4. **0x00001157 <+30>: mov EBP,EAX**
   - **Descrizione:** Questa istruzione sposta il valore del registro `EAX` (88) nel registro `EBP`.
   - **Scopo:** Conservare il risultato della somma in un registro diverso (`EBP`). Questo permette di mantenere il risultato accessibile per ulteriori confronti o operazioni senza alterare `EAX`.
5. **0x0000115a <+33>: cmp EBP,0xa**
   - **Descrizione:** Confronta il valore del registro `EBP` (88) con il valore esadecimale `0xa` (10 in decimale). Questa operazione imposta i flag del processore in base al risultato della sottrazione implicita (`EBP - 10`).
   - **Scopo:** Determinare se il valore contenuto in `EBP` è maggiore, uguale o minore di 10. Questa operazione è cruciale per decidere il flusso di esecuzione successivo.
6. **0x0000115e <+37>: jge 0x1176 <main+61>**
   - **Descrizione:** Esegue un salto all'indirizzo `0x1176` se il risultato del confronto precedente è maggiore o uguale a zero (cioè, se `EBP` è maggiore o uguale a 10).
   - **Scopo:** Questa istruzione condizionale modifica il flusso del programma in base al risultato del confronto. Se `EBP` è 88 (che è maggiore di 10), il controllo del programma salta all'indirizzo `0x1176`.
7. **0x0000116a <+49>: mov eax,0x0**
   - **Descrizione:** Sposta il valore esadecimale `0x0` (0 in decimale) nel registro `EAX`.
   - **Scopo:** Inizializza nuovamente `EAX`, questa volta con il valore 0. Questo può essere fatto per preparare `EAX` per un nuovo utilizzo o per resettare il suo valore.
8. **0x0000116f <+54>: call 0x1030 printf@plt**
   - **Descrizione:** Chiama la funzione `printf` all'indirizzo `0x1030` nel Procedure Linkage Table (PLT).

- **Scopo:** Invoca la funzione `printf`, che è tipicamente utilizzata per stampare un messaggio di output. Questa chiamata è essenziale per fornire feedback all'utente o per scopi di debug.

Analisi dei Flag

In Assembly, i flag del processore sono indicatori che segnalano il risultato delle operazioni aritmetiche e logiche. I flag principali coinvolti sono:

- **ZF (Zero Flag):** Viene impostato se il risultato di un'operazione è zero.
- **CF (Carry Flag):** Viene impostato se c'è un riporto o un prestito fuori dal bit più significativo durante un'operazione aritmetica.

Analisi Complessiva

Il programma Assembly inizia con l'inizializzazione dei registri `EAX` e `EDX` con i valori 32 e 56 rispettivamente. Successivamente, i valori vengono sommati e il risultato (88) viene memorizzato in `EAX`, poi copiato in `EBP`. Viene quindi eseguito un confronto tra `EBP` (88) e 10. Se `EBP` è maggiore o uguale a 10, il programma salta all'indirizzo `0x1176`. Se il salto non viene eseguito, `EAX` viene impostato a 0 e viene chiamata la funzione `printf`.

Questo codice verifica se la somma di 32 e 56 è maggiore o uguale a 10 e, se sì, modifica il flusso del programma; altrimenti, esegue una stampa tramite `printf`.

Conclusione

Questa esercitazione ha permesso di comprendere meglio il funzionamento delle istruzioni in Assembly e come queste interagiscono tra di loro.