**S10 L3**

**Assembly x86**

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>

RISOLUZIONE

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

-**0x00001141** è l'indirizzo di memoria dove questa istruzione è memorizzata. In questo caso, l'istruzione mov EAX, 0x20 è memorizzata all'indirizzo esadecimale 0x00001141.

- **<+8>:** Questo indica l'offset relativo all'inizio di una funzione o di un blocco di codice. In altre parole, questa istruzione si trova 8 byte dopo l'inizio della funzione o del blocco di codice corrente. Questo tipo di annotazione è spesso utilizzato nei listati di codice generati dai debugger o dai disassemblatori per mostrare la posizione relativa delle istruzioni all'interno di una funzione.

**Carica** il valore esadecimale 0x20 (che in numero decimale corrisponde a 32) nel registro **EAX**. Questo registro è spesso utilizzato per operazioni aritmetiche e di logica.

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

**Carica** il valore esadecimale 0x38 (che in numero decimale corrisponde a 56) nel registro **EDX**. Come **EAX**, anche **EDX** è un registro di uso generale.

0x00001155 <+28>: add EAX,EDX

**Somma** il valore del registro **EDX** (56) al valore del registro **EAX** (32), e memorizza il risultato (88) nel registro **EAX**.

0x00001157 <+30>: mov EBP, EAX

**Copia** il valore corrente del registro **EAX** (88) nel registro **EBP. EBP** è comunemente usato come puntatore di base per il frame dello stack, ma può essere utilizzato anche come registro di uso generale.

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

**Confronta** il valore del registro **EBP** (88) con il valore esadecimale **0xa** (10 in decimale). Questa istruzione effettua una sottrazione implicita di 0xa da **EBP**, impostando i flag di condizione basati sul risultato.

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

**Salta** all'indirizzo 0x1176 (etichetta main+61) se il valore nel registro **EBP** è maggiore o uguale a **0xa** (10). jge sta per **"jump if greater or equal"** e dipende dai flag impostati dalla precedente istruzione cmp.

0x0000116a <+49>: mov EAX,0x0

**Carica** il valore 0x0 (0 in decimale) nel registro **EAX.** Questa istruzione è comunemente utilizzata per preparare un valore di ritorno.

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

**Chiama** la funzione printf situata all'indirizzo 0x1030. L'indirizzo 0x1030 è l'indirizzo della funzione printf nella **Procedure Linkage Table (PLT)**, che viene utilizzata per gestire le chiamate a funzioni esterne come quelle della libreria standard C.

CONCLUSIONI

Il programma esegue le seguenti operazioni:

1. Carica l’intero 32 nel registro EAX.
2. Carica l’intero 56 nel registro EDX.
3. Somma i valori in EAX e EDX e memorizza il risultato (88) in EAX.
4. Copia il valore di EAX (88) in EBP.
5. Confronta il valore di EBP (88) con 10.
6. Se EBP è maggiore o uguale a 10, salta all'indirizzo 0x1176.
7. Se non salta, carica 0 in EAX.
8. Chiama la funzione printf.

Il programma esegue una semplice operazione aritmetica, confronta il risultato con un valore costante e poi, a seconda del risultato del confronto, decide se saltare a una parte del codice o continuare con una chiamata a printf.

FUNZIONAMENTO DELLE ISTRUZIONI

L'istruzione mov in linguaggio Assembly è una delle istruzioni più fondamentali e utilizzate. La sua funzione principale è copiare un valore da una posizione a un'altra. Può copiare valori tra registri, tra un registro e una locazione di memoria, o tra un registro e un valore immediato. È utilizzata in una vasta gamma di operazioni, dalla configurazione iniziale dei registri alla manipolazione dei dati in memoria

L’ISTRUZIONE add in linguaggio Assembly è utilizzata per sommare due operandi. Il risultato della somma viene memorizzato nell'operando di destinazione. Questa istruzione può operare su registri, valori immediati e locazioni di memoria.

L'istruzione cmp in linguaggio Assembly è utilizzata per confrontare due operandi. Non modifica nessuno degli operandi, ma aggiorna i flag nei registri di stato della CPU per riflettere il risultato del confronto. Questa istruzione è molto simile a un'istruzione di sottrazione (sub), ma senza memorizzare il risultato della sottrazione. L'istruzione **cmp e add** aggiornano diversi flag nei registri di stato della CPU, utili per prendere decisioni successive:

* **ZF (Zero Flag)**: Viene impostato se il risultato della sottrazione implicita è zero.
* **SF (Sign Flag)**: Viene impostato se il risultato della sottrazione implicita è negativo.
* **OF (Overflow Flag)**: Viene impostato se c'è un overflow con segno.
* **CF (Carry Flag)**: Viene impostato se c'è un prestito durante la sottrazione.
* **AF (Adjust Flag)**: Utilizzato principalmente per aritmetica binaria in BCD (Binary-Coded Decimal).

L'istruzione jge **(jump if greater or equal)** in linguaggio Assembly è una delle istruzioni di salto condizionale utilizzate per controllare il flusso di esecuzione del programma in base ai risultati di un confronto precedente effettuato con l'istruzione cmp o test. L'istruzione jge esegue un salto a **un'etichetta** specificata se il risultato del confronto indica che il primo operando è maggiore o uguale al secondo operando, interpretando i valori come numeri con segno.

In Assembly, **un'etichetta (o "label" in inglese)** è un identificatore utilizzato per marcare una posizione specifica nel codice. Le etichette sono fondamentali per il controllo del flusso del programma, in quanto consentono di specificare destinazioni per salti condizionali e incondizionati, chiamate a subroutine e altre istruzioni per il controllo del flusso.

L'istruzione call in linguaggio Assembly è utilizzata per chiamare una subroutine o funzione (in questo caso la funzione chiamata è printf). Questa istruzione salva l'indirizzo di ritorno (l'indirizzo dell'istruzione successiva alla call) sullo stack e poi salta all'indirizzo specificato dalla subroutine.

Perché è importante sapere il numero di byte dopo un'istruzione?

E’ importante per diverse ragioni:

1. **Debugging**: Quando si sta esaminando il codice assembly in un debugger, l'offset può aiutare a identificare rapidamente l'istruzione attualmente eseguita e può essere utile nel tracciare il flusso di esecuzione del programma.
2. **Ottimizzazione del codice**: Quando si ottimizza il codice, avere familiarità con gli offset può aiutare a capire come le istruzioni sono organizzate in memoria e come possono essere riorganizzate per ottimizzare il tempo di esecuzione o la dimensione del codice.
3. **Analisi del codice generato dal compilatore**: Capire l'offset può essere utile per analizzare il codice generato dal compilatore e comprendere come il compilatore ottimizza e organizza il codice sorgente in codice macchina.
4. **Reverse engineering**: Quando si effettua reverse engineering su un programma, conoscere l'offset delle istruzioni può essere utile per tracciare la logica del programma e identificare eventuali vulnerabilità o comportamenti interessanti.

Per concludere, l'offset fornisce un'informazione aggiuntiva sulle istruzioni del codice assembly che può essere sfruttata in varie fasi dello sviluppo del software e nell'analisi dei programmi eseguibili.

Che cosa sono i flag di registro?

Nel linguaggio assembly, i **"flag di registro"** sono i bit speciali di stato che sono contenuti in un registro specifico (solitamente chiamato registro di flag) nel processore. Questi bit rappresentano lo stato corrente del processore dopo l'esecuzione di un'istruzione. I flag di registro sono fondamentali per il controllo del flusso di esecuzione del programma e per la gestione delle operazioni aritmetiche e logiche.

I flag di registro comuni includono:

1. **Zero Flag (ZF)**: Impostato se il risultato di un'operazione è zero.
2. **Carry Flag (CF)**: Usato per segnalare il riporto o il prestito da un'operazione aritmetica.
3. **Overflow Flag (OF)**: Indica se un'operazione aritmetica ha prodotto un risultato che supera la capacità di rappresentazione del registro.
4. **Sign Flag (SF)**: Indica il segno del risultato di un'operazione aritmetica (positivo o negativo).
5. **Parity Flag (PF)**: Indica se il numero di bit impostati nel risultato di un'operazione è pari o dispari.
6. **Direction Flag (DF)**: Utilizzato per specificare la direzione in cui vengono elaborati gli elementi in una stringa di dati.
7. **Interrupt Flag (IF)**: Abilita o disabilita le interruzioni hardware.
8. **Trap Flag (TF)**: Utilizzato per attivare la modalità di debug, consentendo l'esecuzione di istruzioni singole.

E’ importante ricordare che questi flag sono modificati automaticamente dal processore in base all'esecuzione delle istruzioni e possono essere utilizzati dalle istruzioni condizionali per prendere decisioni durante l'esecuzione del programma.