Report Dettagliato sull'Esercitazione

Introduzione

In questa esercitazione, ho affrontato il compito di ricostruire le istruzioni originali in C a partire dal codice assembly fornito. L'obiettivo era di comprendere il funzionamento delle istruzioni assembly e tradurle in un linguaggio di alto livello, mantenendo la struttura e la logica del programma originale.

Analisi del Codice Assembly

Primo Blocco di Codice

```
push   %ebp
mov    %esp,%ebp
sub    $0x8,%esp
call   80483e9 <bar>
leave
ret
```

Spiegazione:

- `push %ebp`: Salva il valore del frame pointer corrente.
- `mov %esp,%ebp`: Imposta il frame pointer (ebp) al valore dello stack pointer (esp).
- `sub $0x8,%esp`: Riserva 8 byte di spazio sullo stack per le variabili locali.
- `call 80483e9 <bar>`: Chiama la funzione `bar`.
- `leave`: Ripristina il frame pointer originale.
- `ret`: Ritorna dalla funzione.

Traduzione in C:

```
void foo() {
    bar();
}
```

Secondo Blocco di Codice

```
push   %ebp
mov    %esp,%ebp
sub    $0x8,%esp
call   80483fb <baz>
call   8048400 <quux>
leave
ret
```

Spiegazione:

- `push %ebp`: Salva il valore del frame pointer corrente.
- `mov %esp,%ebp`: Imposta il frame pointer (ebp) al valore dello stack pointer (esp).
- `sub $0x8,%esp`: Riserva 8 byte di spazio sullo stack per le variabili locali.
- `call 80483fb <baz>`: Chiama la funzione `baz`.
- `call 8048400 <quux>`: Chiama la funzione `quux`.
- `leave`: Ripristina il frame pointer originale.

- `ret`: Ritorna dalla funzione.

Traduzione in C:

```
void foo() {
    baz();
    quux();
}
```

Terzo Blocco di Codice

```
push   %ebp
mov    %esp,%ebp
pop    %ebp
ret
```

Spiegazione:

- `push %ebp`: Salva il valore del frame pointer corrente.
- `mov %esp,%ebp`: Imposta il frame pointer (ebp) al valore dello stack pointer (esp).
- `pop %ebp`: Ripristina il valore originale del frame pointer.
- `ret`: Ritorna dalla funzione.

Traduzione in C:

```
void foo() {
    // Nessuna operazione
}
```

Quarto Blocco di Codice

```
push   %ebp
mov    %esp,%ebp
mov    $0x0,%eax
movl   $0x1,(%eax)
pop    %ebp
ret
```

Spiegazione:

- `push %ebp`: Salva il valore del frame pointer corrente.
- `mov %esp,%ebp`: Imposta il frame pointer (ebp) al valore dello stack pointer (esp).
- `mov $0x0,%eax`: Imposta il registro eax a 0.
- `movl $0x1,(%eax)`: Scrive il valore 1 nella memoria all'indirizzo puntato da eax (che è 0, quindi questo causerà un segfault).
- `pop %ebp`: Ripristina il valore originale del frame pointer.
- `ret`: Ritorna dalla funzione.

Traduzione in C:

```
void foo() {
    int *ptr = 0;
    *ptr = 1; // Questo causerà un segmentation fault
}
```

Quinto Blocco di Codice

```
push   %ebp
mov    %esp,%ebp
and    $0xfffffff0,%esp
call   80483dc <foo>
mov    $0x0,%eax
leave
ret
```

Spiegazione:

- `push %ebp`: Salva il valore del frame pointer corrente.
- `mov %esp,%ebp`: Imposta il frame pointer (ebp) al valore dello stack pointer (esp).
- `and $0xfffffff0,%esp`: Allinea lo stack pointer a un indirizzo a 16 byte.
- `call 80483dc <foo>`: Chiama la funzione `foo`.
- `mov $0x0,%eax`: Imposta il valore di ritorno a 0.
- `leave`: Ripristina il frame pointer originale.
- `ret`: Ritorna dalla funzione.

Traduzione in C:

```
void foo() {
    foo();
    return 0;
}
```

Codice C Completo

Unendo tutte le traduzioni, otteniamo il seguente programma in C:

```
#include <stdio.h>

void foo();
void bar();
void baz();
void quux();

void foo() {
    bar();
}

void bar() {
    baz();
    quux();
}

void baz() {
    // do nothing
}

void quux() {
    *(int *)(0) = 1; // Questo causerà un segmentation fault
}

int main() {
    foo();
    return 0;
}
```

Conclusione

In questa esercitazione, ho tradotto vari blocchi di codice assembly in codice C. Questo processo mi ha permesso di comprendere meglio la corrispondenza tra le istruzioni assembly e le operazioni di alto livello in C. Ho anche appreso come identificare le funzioni e le chiamate di funzione nel codice assembly, oltre a riconoscere le operazioni di gestione dello stack e delle variabili locali. Questo esercizio è stato fondamentale per migliorare la mia comprensione dell'assembly e delle sue traduzioni in linguaggio di alto livello.