Introduzione Assembly

Leonardo Bizzoni

April 2, 2023

Un assembler legge un singolo **file assembly** e produce un object file che contiene **istruzioni macchina**.

È possibile scrivere un programma assembly in file distinti, chiamati **moduli**, che vengono assemblati **singolarmente**. Un programma assembly può utilizzare anche delle procedure provvedute da **librerie** preassemblate.

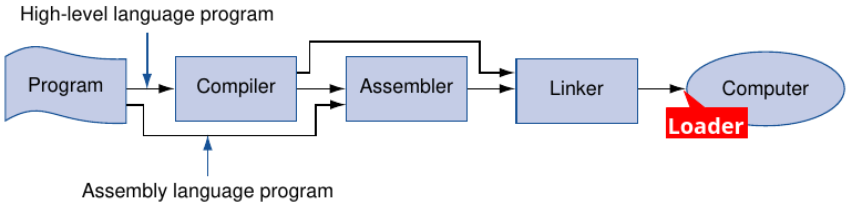
Un modulo non può venire eseguito se contiene delle **unresolved reference** a label in altri object file o librerie. Il linker ha il compito di risolvere questo tipo di reference unendo tutti gli object file e librerie in un unico file eseguibile.

Nomi che cominciano con un '.', come *.data, .text* o *.globl* sono dette **direttive**. Le direttive dicono all'assembler come tradurre le successive operazioni/valori ma non producono alcuna istruzione macchina.

Nomi seguiti da ':' sono **label** ovvero dei nomi associati ad un'istruzione.

Assembly ha 2 ruoli principali:

- output generato da un **compiler** il cui compito è quello di tradurre un linguaggio di *alto livello* direttamente in linguaggio macchina o in assembly
- effettivo linguaggio di programmazione di *basso livello*



Lo svantaggio di scrivere programmi direttamente in assembly è che assembly è legato all'architettura hardware e deve essere riscritto completamente per eseguirlo su un'altra architettura (*Un programma assembly scritto per MIPS non funziona su ARM*).

Un compiler può produrre istruzioni macchina senza bisogno di un assembler. Questo tipo di compilazione è tipicamente molto più rapida ma richiede un compiler molto più complesso che deve svolgere anche il compito di un assembler.