Lezione 2

5–6 ottobre 1999

Argomenti trattati

- Computer e programmazione, hardware e software.
- La macchina di Von Neumann.
- Linguaggio Macchina.
- Linguaggi ad alto livello.
- Compilatori.
- Editor, Linker, Interpreti.

2.1 Computer e programmazione, hardware e software

Se volessimo definire cos'è un computer, potremmo dire che un *computer* è una macchina elettronica *programmabile* al fine di svolgere diverse funzioni. La caratteristica fondamentale che distingue un computer, ad esempio, da una calcolatrice, è appunto la sua *programmabilità*: avendo in ingresso un programma e dei dati, un computer produce risultati secondo il procedimento di trasformazione descritto dal programma.

Equivalentemente, possiamo definire un computer come una macchina in grado di eseguire un processo computazionale sulla base di regole specificate tramite un *programma*. Il programma è costruito a partire da istruzioni elementari che il computer è in grado di comprendere ed eseguire. L'insieme di tali istruzioni è generalmente piuttosto ristretto; la potenza e la vasta applicabilità dei computer è data dalla capacità di operare in maniera estremamente veloce ed accurata. Combinando azioni elementari in lunghe sequenze di istruzioni è possibile far eseguire a un computer compiti complessi.

La *programmazione* è l'attività che consiste nell'organizzare istruzioni elementari, direttamente comprensibili dall'esecutore, in strutture complesse, i *programmi*, al fine di svolgere determinati compiti. Quando introdurremo il concetto di *compilatore* vedremo che, fortunatamente, non è necessario scrivere programmi nel linguaggio, estremamente povero, comprensibile dal processore di un computer, ma possiamo utilizzare linguaggi ad alto livello, comprensibili da esecutori *astratti*.

Con i termini *hardware* e *software* indichiamo le due "anime" di un computer: l'hardware è la parte fisica, costituita da un insieme di circuiti opportunamente connessi e da vari dispositivi, il software è l'insieme di tutti i programmi.

Hardware

La configurazione dell'hardware può variare notevolmente da un elaboratore all'altro, sia per quanto riguarda i dispositivi presenti, sia per le loro caratteristiche. Tuttavia, senza addentrarci in particolari che verranno approfonditi in altri corsi, possiamo riconoscere alcuni elementi fondamentali comuni.

- Il *processore* o *unità centrale* o, ancora, CPU (dall'inglese *Central Processing Unit*). È la parte che esegue effettivamente l'elaborazione.

- La *memoria centrale*.
  Contiene il programma (o i programmi) in esecuzione e i dati (o parte dei dati) su cui esso opera. La memoria centrale ha capacità limitata (oggi vengono venduti personal computer con una memoria centrale dell'ordine di un centinaio di Megabytes), ma permette di reperire i dati abbastanza velocemente. Infatti la memoria centrale è una memoria ad accesso diretto (detta anche RAM, dall'inglese *Random Access Memory*), cioè una memoria nella quale un dato può essere reperito conoscendone la posizione. La memoria RAM è una memoria a lettura e scrittura, cioè in essa è possibile leggere o scrivere dati. L'informazione nella memoria RAM viene mantenuta solo in presenza di alimentazione. La memoria centrale di un computer contiene di solito anche un'area, detta ROM (dall'inglese *Read Only Memory*), a sola lettura, di dimensioni piuttosto limitate, che mantiene l'informazione anche in assenza di alimentazione. Quest'area è utilizzata per contenere informazioni necessarie, al momento dell'accensione, all'avvio del computer.

  La memoria centrale di un elaboratore è suddivisa in *celle* (dette anche *parole* o *locazioni*) tutte della stessa dimensione (ad esempio di 2 byte). Ogni cella è identificabile tramite un numero, detto *indirizzo*. Il processore può effettuare un'operazione di lettura o scrittura in una cella di memoria, specificandone l'indirizzo. Si osservi che, quando si effettua la scrittura di un dato in una cella di memoria, il valore contenuto precedentemente nella cella viene perso. Il tempo d'accesso a una cella è costante ed indipendente dalla sua posizione.

- La *memoria di massa*.
  Al contrario della memoria centrale, la memoria di massa è in grado di memorizzare grandi quantità di informazione (gli hard disk dei personal computer odierni hanno capacità di diversi Gigabytes). Inoltre l'informazione viene memorizzata in maniera permanente, cioè non viene persa quando manca l'alimentazione. Il tempo necessario per accedere alla memoria di massa risulta però di gran lunga superiore rispetto a quello necessario per accedere alla memoria centrale. L'informazione contenuta nella memoria di massa è organizzata in *archivi* o *file*.

- Le *periferiche*.
  Sono dispositivi che permettono la comunicazione tra il computer e l'ambiente esterno ad esso. Esempi di periferiche sono la tastiera, il monitor, il mouse, il modem, la stampante, ecc.

- Il *bus*.
  È essenzialmente un insieme di cavi, utilizzati per collegare tra loro i componenti indicati sopra.

Software

I programmi utilizzabili su un calcolatore possono svolgere svariati compiti. Possiamo suddividere l'insieme dei programmi nei seguenti tre gruppi fondamentali.

- Il *Sistema Operativo*.
  Ha il compito di controllare e coordinare l'uso di tutte le risorse della macchina. Alcune delle funzionalità fondamentali svolte dal sistema operativo sono:
  - ricevere ed eseguire i comandi impartiti dall'utente tramite la *shell*;
  - avviare l'esecuzione dei programmi;
  - rendere disponibili le risorse (memoria, rete, periferiche, ecc.);
  - permettere la condivisione ottimale delle risorse tra piú utenti.

  Il sistema operativo si occupa tra l'altro della gestione e dell'accesso al *file system*, cioè all'insieme di tutti i file presenti sulla memoria di massa.

- *Utilities* e *software di base*.
  Sono programmi che permettono attività legate alla gestione della macchina e dei file (esempio programmi per la copia di file, programmi compressori, antivirus, ecc.) o strumenti di sviluppo (editor, compilatori, interpreti, linker, ecc.).

- *Programmi applicativi*.
  Sono programmi che svolgono direttamente le funzionalità che interessano all'utente finale.

2.2 La macchina di Von Neumann

Abbiamo visto quali sono i componenti fondamentali dell'hardware di un elaboratore. Il particolare, il "cuore" dell'elaboratore è costituito da due componenti: la *memoria*, contenente il programma da eseguire e i dati da esso utilizzati, e il *processore*, cioè l'esecutore. Questo modello, proposto da *Von Neumann* nel 1946, è utilizzato per tutti gli elaboratori convenzionali.

Il processore opera ripetendo ciclicamente le seguenti operazioni:

- Preleva dalla memoria la prossima istruzione da eseguire (fase di *Fetch*).
- Interpreta l'istruzione, cioè ne riconosce il significato (fase di *Decode*).
- Esegue le operazioni corrispondenti all'istruzione (fase di *Execute*).

Possiamo subito osservare che le istruzioni da eseguire, e dunque i programmi, devono essere scritti in un linguaggio comprensibile dal processore, che viene detto *linguaggio macchina*. Inoltre, il processore non è in grado di elaborare direttamente i dati contenuti nella memoria centrale, ma può operare solo su dati che si trovano all'interno di appositi registri, contenuti nel processore stesso. Pertanto, per effettuare un'operazione su dati contenuti nella memoria, è necessario traferire i dati dalla memoria nei registri del processore, effettuare l'operazione utilizzando i registri, trasferire il risultato nella memoria.

Per chiarire meglio questi aspetti, presentiamo alcuni esempi. Supponiamo di disporre di un processore, dotato di due registri `R1` e `R2`. Vogliamo calcolare la somma dei valori contenuti in due locazioni di memoria a e b, e porre il risultato in una locazione c.

Le operazioni da compiere saranno:

1. Copia il contenuto della cella di memoria a nel registro `R1`.
2. Copia il contenuto della cella di memoria b nel registro `R2`.
3. Somma il contenuto dei registri `R1` e `R2` e poni il risultato in `R1`.
4. Copia il contenuto del registro `R1` nella cella di memoria c.

Le operazioni precedenti corrispondono a istruzioni elementari del nostro processore, in particolare abbiamo:

- *Istruzioni di trasferimento dati tra memoria e processore.*
  L'istruzione che permette di "caricare" in un registro `R` il valore contenuto in una cella di memoria di indirizzo x verrà indicata con `LOAD R, x`.
  L'istruzione che permette di "scaricare" o, meglio, copiare nella cella di indirizzo x il valore contenuto nel registro `R` verrà indicata con `STORE R, y`.

- *Istruzioni aritmetico/logiche.*
  Permettono di effettuare operazioni aritmetiche o logiche tra i dati contenuti dei registri, e lasciano il risultato in un registro. Ad esempio, l'istruzione `ADD R',R''` calcola la somma dei valori contenuti in `R'` e `R''`, ponendo il risultato nel primo registro specificato, cioè `R'`. Analogamente, nei nostri esempi, considereremo istruzioni per il calcolo della sottrazione, del prodotto e della divisione intera, indicate rispettivamente con `SUB`, `MUL` e `DIV`.

- *Istruzioni di controllo e di salto.*
  Permettono di cambiare l'ordine con cui vengono eseguite le istruzioni. Verranno descritte successivamente.

I nomi, come `LOAD`, `STORE`, `ADD`, ecc., che abbiamo utilizzato per indicare le istruzioni, sono codici mnemonici che ci aiutano a ricordare quali siano le operazioni svolte (cioè la *semantica* delle istruzioni). In realtà nel linguaggio macchina, cioè nel linguaggio del processore, a ciascuno di questi nomi corrisponde una sequenza di cifre binarie. Pertanto, per scrivere un programma direttamente il linguaggio macchina è necessario utilizzare, al posto dei codici mnemonici, le cifre binarie corrisponenti. Ricordiamo, inoltre, che questo è solo un esempio. Ogni processore ha un proprio formato di istruzioni macchina. Tuttavia, le classi di istruzioni presenti, sebbene molto piú articolate, sono analoghe per quasi tutti i processori.

Il linguaggio costituito dai codici mnemonici associati alle istruzioni macchina viene detto *linguaggio assembler*. Chiaramente, ogni processore ha un proprio linguaggio assembler.

Utilizzando i codici mnemonici appena introdotti, l'assegnamento alla locazione c della somma dei contenuti delle locazioni di memoria a e b, può essere effettuato mediante le seguenti istruzioni:

```
LOAD  R1, a
LOAD  R2, b
ADD   R1,R2
STORE R1, c
```

(In realtà al posto di `a`, `b` e `c`, andranno scritti gli indirizzi di memoria corrispondenti.)

Le istruzioni di controllo e di salto permettono di modificare l'ordine, strettamente sequenziale, con cui vengono eseguite solitamente le istruzioni di un programma. È possibile associare a un'istruzione un'etichetta, scrivendo ad esempio

```
et: LOAD R1, a
```

L'istruzione `JUMP et` fa proseguire l'esecuzione dall'istruzione preceduta dall'etichetta `et`. L'istruzione `JZERO R,et`, invece, permette di decidere quale sia la prossima istruzione da eseguire, in base al valore contenuto nel registro `R`: in particolare, se `R` contiene zero, l'esecuzione prosegue dall'istruzione preceduta dall'etichetta `et`, se invece `R` contiene un valore diverso da zero, l'esecuzione prosegue normalmente con l'istruzione successiva.

Ad esempio, l'esecuzione del codice:

```
        LOAD  R1, 30
        LOAD  R2, 31
        SUB   R1, R2
        JZERO R1, alfa
        LOAD  R1, 30
        ADD   R1, R2
alfa:   STORE R1, 40
```

ha come effetto quello di porre nella cella 40 la somma dei valori contenuti nelle celle 30 e 31, nel caso essi siano diversi, e di porre nella cella 40 il valore zero, nel caso essi siano uguali.

Presentiamo ora un esempio piú articolato, quello del calcolo del massimo comun divisore tra due numeri, mediante l'algoritmo di Euclide che, ricordiamo, è il seguente:

1. Siano x e y i due numeri.
2. Calcola il resto della divisione di x per y.
3. Se il resto è diverso da zero, ricomincia dal passo 2
   utilizzando come x il valore attuale di y, e come y il valore del resto,
   altrimenti prosegui con il passo successivo.
4. Il massimo comun divisore è uguale al valore attuale di y.

Supponiamo che i due numeri si trovino agli indirizzi di memoria 101 e 102, e che il risultato vada posto all'indirizzo 103. Per semplicità supponiamo inoltre di potere cambiare i contenuti delle locazioni 101 e 102 durante il calcolo (questo comporta la perdita dei dati iniziali).

All'inizio dell'algoritmo di Euclide, i numeri di cui si vuole calcolare il massimo comun divisore vengono indicati con x e y. Pertanto, x corrisponderà alla locazione 101 e y alla locazione 102. Per effettuare le operazioni, è necessario trasferire i numeri nei registri; dunque, le prime istruzioni saranno:

```
LOAD  R1, 101
LOAD  R2, 102
```

Il passo successivo è il calcolo del resto della divisione di x per y. Si osservi che, indicando con il simbolo "/" la divisione intera, il resto è dato dal risultato dell'espressione $x - (x/y) * y$.

Per il calcolo di tale espressione possiamo eseguire i seguenti passi:

- Calcoliamo x/y, con l'istruzione `DIV R1, R2`. Questa istruzione lascia il risultato in `R1` (che dunque non conterrà piú il valore di x).
- Calcoliamo $(x/y) * y$, moltiplicando il risultato precedentemente ottenuto per il valore di y, con l'istruzione `MUL R1, R2`. A questo punto `R1` contiene il risultato dell'espressione $(x/y) * y$, che andrà sottratto da x.
- Calcoliamo $x-(x/y)*y$. Per fare questo carichiamo in `R2` il valore di x, utilizzando l'istruzione `LOAD R2, 101`, e sottraiamo da `R2` il contenuto di `R1`, utilizzando l'istruzione `SUB R2, R1`. Si osservi che, poiché il risultato viene lasciato nel primo registro specificato, esso si troverà in `R2`.

A questo punto `R2` contiene il resto della divisione. Possiamo quindi espandere il passo 3 dell'algoritmo di Euclide. Se il registro `R2` contiene zero, occorre ricominciare dal passo 2, dopo avere modificato i valori di x e y; se il registro `R2` contiene zero, invece, si prosegue con il passo 4. Utilizziamo l'istruzione di salto condizionato `JZERO R2, fine` per trattare il caso di resto zero. Dovremo poi scrivere un blocco di istruzioni, che inizi con l'etichetta `fine`, per rappresentare il passo 4 dell'algoritmo.

Quando il resto è diverso da zero, prima di ripetere dal passo 2, occorre spostare il valore attuale di y in x (cosa che può essere realizzata con le due istruzioni `LOAD R1, 102` e `STORE R1, 101`), e copiare il valore del resto, che si trova in `R2`, in y, mediante l'istruzione `STORE R2, 102`. Si noti che a questo punto `R1` e `R2` contengono i nuovi valori di x e y. Dunque si può riprendere dall'istruzione `DIV R1, R2`, scritta in precedenza. Per fare questo, introduciamo l'istruzione `JUMP alfa`, dove l'etichetta `alfa` verrà associata all'istruzione di divisione.

Infine, dobbiamo scrivere le istruzioni corrispondenti al passo 4, il cui compito è quello di lasciare il massimo comun divisore, che secondo l'algoritmo di Euclide si trova in y cioè nella cella 102, nella cella 103. Alla prima di queste istruzioni deve essere associata l'etichetta `fine`. Il codice è dunque

```
fine: LOAD  R1, 102
      STORE R1, 103
```

Ecco il codice completo:

```
      LOAD  R1, 101
      LOAD  R2, 102
alfa: DIV   R1, R2
      MUL   R1, R2
      LOAD  R2, 101
      SUB   R2, R1
      JZERO R2, fine
      LOAD  R1, 102
      STORE R1, 101
      STORE R2, 102
      JUMP  alfa
fine: LOAD  R1, 102
      STORE R1, 103
```

Per comprendere meglio il comportamento del codice presentato sopra, si consiglia di utilizzare carta e penna per simularne l'esecuzione da parte della macchina.

2.3 Dal linguaggio macchina ai linguaggi ad alto livello

Gli esempi presentati mostrano che, utilizzando il linguaggio macchina, anche il calcolo di semplici espressioni aritmetiche deve essere trasformato in una lunga e poco comprensibile sequenza di istruzioni. La programmazione in linguaggio macchina, praticata sino agli inizi degli anni '50, presenta dunque notevoli svantaggi:

- È necessario conoscere i dettagli dell'architettura del processore utilizzato e il relativo linguaggio (che è formato da una sequenza poco leggibile di codici).
- Poiché ogni processore ha un proprio linguaggio macchina, risulta impossibile trasportare i programmi da una macchina ad una differente.
- Il programmatore deve conoscere in dettaglio tutte le caratteristiche della macchina che utilizza.
- Il programmatore si specializza nell'uso di "trucchi" legati alle caratteristiche specifiche della macchina. In questo modo i programmi risultano pressoché incomprensibili e difficilmente modificabili, persino dalla stessa persona che li ha scritti.
- Leggendo il programma è difficile comprenderne la struttura e individuare eventuali errori.
- È necessario riscrivere completamente i programmi quando essi vadano trasferiti su una macchina differente.

In sintesi, possiamo osservare che tutta l'attività di programmazione in linguaggio macchina è strettamente dipendente dalle caratteristiche del processore utilizzato, alle quali il programmatore è costretto ad adeguarsi.

Per rendere la programmazione indipendente dalle caratteristiche peculiari della macchina utilizzata, sono stati introdotti i cosidetti *linguaggi ad alto livello*. Essi non sono pensati per essere compresi direttamente da macchine reali, ma da macchine "astratte", in grado di effettuare operazioni piú ad alto livello, rispetto alle operazioni elementari dei processori reali. In questo modo

l'attività di programmazione viene svincolata dalla conoscenza dei dettagli architetturali della macchina utilizzata.

Chiaramente, per poter poi eseguire, su una macchina reale, un programma scritto in linguaggio ad alto livello, è necessario trasformarlo, o meglio *tradurlo*, nel linguaggio della macchina utilizzata. Questa operazione, fortunatamente, può essere effettuata per via automatica, cioè utilizzando appositi programmi che prendono il nome di *compilatori*. Ad esempio, un compilatore Pascal per una macchina X, è un programma che riceve in ingresso un programma P scritto in linguaggio Pascal, e produce in uscita un programma P', equivalente a P, cioè tale che le esecuzioni di P e di P' sugli stessi dati di ingresso producono gli stessi risultati, scritto nel linguaggio della macchina X.

Dunque, un programmatore che conosca il linguaggio Pascal e che disponga di un compilatore Pascal per la macchina X potrà scrivere programmi da eseguire su X, senza conoscerne il linguaggio macchina. Inoltre, disponendo di un compilatore Pascal per un'altra macchina Y, gli stessi programmi potranno essere fatti eseguire da Y, senza doverli riscrivere da zero, come succedeva invece programmando in linguaggio macchina. In pratica, il programmatore scrive il proprio programma facendo riferimento a una macchina astratta, la "macchina Pascal"; grazie al compilatore un programma per la "macchina Pascal" potrà essere trasformato in un programma per una macchina reale. La conoscenza del linguaggio macchina è necessaria solo per costruire il compilatore (o meglio solo una piccola parte del compilatore, quella che genera il codice finale).

Una variante del compilatori sono gli *interpreti*. Un interprete è un programma che simula direttamente una macchina astratta. Anziché effettuare la traduzione di P, un interprete Pascal legge ogni istruzione contenuta nel programma P, ed effettua immediatamente, utilizzando la macchina X, le operazioni corrispondenti all'istruzione letta. In pratica, la traduzione dell'intero programma prima dell'esecuzione, viene sostituita dalla traduzione simultanea, con esecuzione immediata di ciascuna istruzione. Si noti che, dovendo eseguire il programma piú volte, risulta piú vantaggioso, in termini di tempo, utilizzare un compilatore, in quanto la traduzione viene effettuata una volta per tutte.

Segnaliamo inoltre l'esistenza degli *assemblatori*, che si occupano di tradurre in linguaggio macchina i programmi scritti con codici mnemonici corrispondenti alle istruzioni macchina, cioè programmi scritti in un linguaggio *assembler*.

2.4 Strumenti per la stesura dei programmi

Nella fase di preparazione dei programmi, si utilizzano vari strumenti automatici (cioè altri programmi). Il primo di tutti è l'*editor*, un programma che permette di scrivere testi. Utilizzando un editor è possibile scrivere in un file il testo del programma nel linguggio ad alto livello. Tale testo viene chiamato *programma sorgente*.

Il programma sorgente può essere quindi dato in ingresso al compilatore che lo traduce nel *codice oggetto*, scritto in linguaggio macchina.

Viene poi utilizzato uno strumento, detto *linker* (letteralmente *collegatore*) che ha il compito di collegare tra loro i vari moduli che costituiscono lo stesso programma. Infatti, in molti linguaggi (non nel Pascal Standard) è possibile suddividere il programma sorgente su piú file, che vengono compilati separatamente creando diversi file oggetto. Il linker ha la funzione di collegare tra loro questi file. Inoltre, un programma può utilizzare alcune funzioni dette di *libreria*, che vengono messe a disposizione del programmatore che le può richiamare direttamente dai programmi (ad esempio le procedure di lettura e scrittura, o alcune funzioni matematiche). Il codice di queste funzioni, che possono anche essere notevolmente complesse, viene collegato al codice oggetto dal *linker*. Il linker produce un *file* contenente il *codice eseguibile* corrispondente al programma di partenza. Il codice eseguibile può essere a questo punto caricato (da parte del *loader* del sistema operativo) per l'esecuzione.

Durante le fasi di preparazione di un programma si possono verificare vari tipi di errore.

Durante la compilazione si possono riscontrare errori di *sintassi* o di *semantica*, dovuti all'uso scorretto del linguaggio. In questo caso, prima di passare alla fasi successive, è necessario correggere, utilizzando di nuovo l'editor, il testo sorgente.

Una volta che il programma sorgente viene compilato senza errori, si può richiamare il linker. Anche qui si possono verificare svariati tipi di errore, quali la mancanza di uno o piú moduli del programma, o l'utilizzo scorretto di funzioni di libreria. Se vi sono errori è necessario richiamare nuovamente il linker, in taluni casi dopo aver anche corretto con l'editor, e compilato nuovamente, uno o piú moduli del programma sorgente.

Una volta che il linker non ha riscontrato errori, è possibile mandare in esecuzione il codice oggetto. Anche durante l'esecuzione possono verificarsi degli errori, che la interrompono. Ad esempio, se viene incontrata un'operazione di divisione, e il divisore vale zero, l'esecuzione viene interrotta. Si noti, che gli errori di esecuzione possono dipendere dai dati in ingresso: lo stesso programma, per alcuni dati può terminare correttamente la propria esecuzione, mentre per altri potrebbe interrompersi, riscontrando un errore.

L'assenza di errori in esecuzione non implica che il programma sia corretto. Infatti il programma potrebbe produrre risultati diversi da quelli aspettati, cioè svolgere una funzione diversa da quella per cui è stato creato. La fase di verifica e di testing, che ha come obiettivo quello di verificare che il programma sia corretto rispetto alle specifiche sulla base delle quali è stato costruito, è una delle fasi piú difficili e delicate. Molte volte i comportamenti anomali si riscontrano solo per particolari valori di ingresso. Inoltre, già in programmi di poche decine di righe, una volta che si individui un errore, può essere difficile risarlire alla causa. Questa fase viene chiamata di *debugging*. Uno strumento utile in questa fase è il *debugger* che permette di osservare passo passo l'andamento dell'esecuzione di un programma.

Una volta che un programma è corretto e funzionante, può ancora essere soggetto a modifiche, ad esempio per aggiungere nuove funzionalità, oltre a quelle per cui era stato inizialmente progettato. Questa fase prende il nome di *manutenzione* e, spesso, è notevolmente complessa. Per ridurre i costi di manutenzione è necessario curare particolamente la stesura iniziale dei programmi in maniera che risultino ben documentati e leggibili. Questo faciliterà, successivamente, la comprensione del significato e del funzionamento del codice.

Molti compilatori per personal computer forniscono ambienti in cui gli strumenti sopra menzionati, editor, compilatore, linker e, talvolta, debugger, sono tra loro integrati. In questo caso, una volta terminata la scrittura del programma con l'editor, è possibile mediante un solo comando richiedere la compilazione, il "linkaggio" e l'esecuzione del programma stesso.