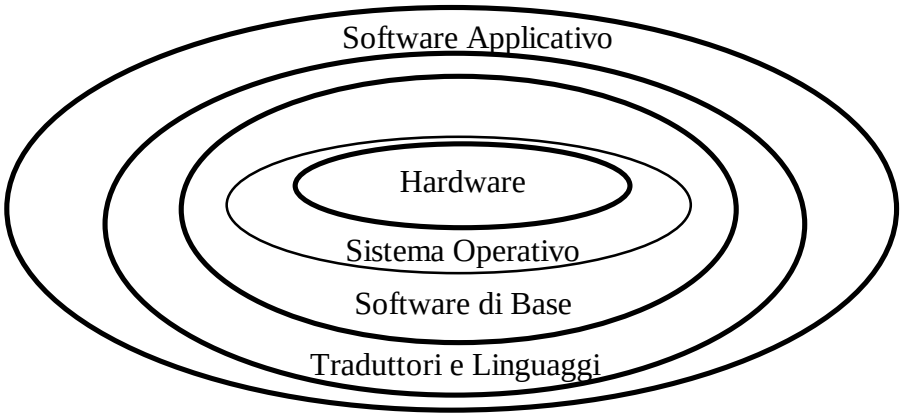
Il software: di base e applicativo

L'HardWare (monitor, tastiera, circuiti, stampante, …) è il nucleo fondamentale del calcolatore ma da solo non serve a nulla. Bisogna utilizzare il software per poterlo fare funzionare. Il *SoftWare di base* è il complesso di programmi che consente di gestire efficacemente l'HW e che viene venduto insieme all'HW essendo di solito prodotto da chi realizza l'HW. Quando acquistiamo un PC non compriamo solo HW ma anche un insieme di programmi già memorizzati nelle memorie permanenti (ROM) o nelle memorie di massa.

Il sistema mette a disposizione un insieme di funzionalità generalizzate. Non ci interessa quali sono i programmi e chi li ha fatti ma ci interessa sapere come utilizzare tali risorse e funzionalità per programmare la specifica soluzione di ogni problema. Molti testi riportano almeno una differenziazione all'interno del SW di base individuando due strati importanti. Quello più interno, fatto dall'insieme dei programmi atti a gestire le risorse HW della macchina, prende il nome di *sistema operativo*. Per esempio l'HW è in grado soltanto di leggere singolarmente il codice del carattere che in un certo istante è stato premuto da tastiera e questo codice è espresso sotto forma di riga e di colonna occupata da quel tasto sulla tastiera. Bene vi sono dei programmi che traducono quel codice (numero di riga e colonna) nell'equivalente codice ASCII del carattere; tale programma è uno di quelli che viene venduto insieme alla macchina e fa parte del sistema operativo. Analogo discorso vale per l'insieme dei programmi che gestiscono il *file system* cioè il sistema di tutti i file, di tutti i programmi, di tutto l'insieme dei dati che sono contenuti nella memoria di massa del sistema: floppy disk, hard disk, etc.

Il sistema operativo più l'HW mettono a disposizione dello strato di SW di base più esterno una *macchina virtuale* come se fosse un HW più potente. *Virtuale* perché è come se a questo livello ci fosse una macchina diversa, più ricca, più dotata di funzionalità dell'HW soltanto.



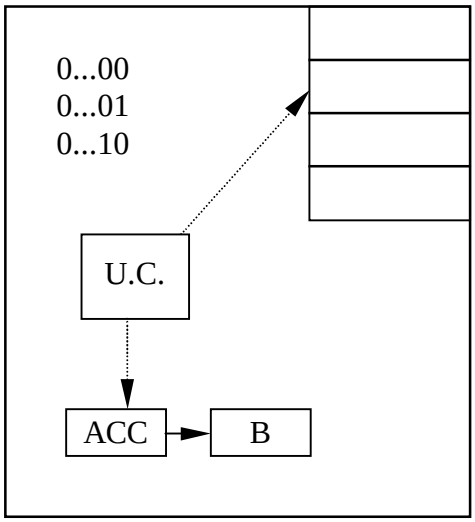
Nel software di base è anche incluso il BIOS (basic input/output system)utilizzato dalla CPU per avviare il sistema dopo averlo acceso. Il BIOS gestisce anche il flusso dati fra il sistema operativo del computer e i dispositivi (le periferiche) come hard disk, schede video, tastiera, mouse e stampanti. Il BIOS è parte integrante del computer. Invece il sistema operativo può essere preinstallato dal fornitore o installato dall'utente. Nel software di base rientra anche un insieme di funzionalità che costituiscono il complesso dei *traduttori* e dei *linguaggi*.

Lo strato più esterno è invece costituito dai *software applicativi*. Un programma applicativo è un programma designato per realizzare una funzione specifica direttamente per l'utente o, in alcuni casi, per altri programmi applicativi. Esempi di software applicativi includono word processor (ossia editori di testo grafici), database, browser Web, strumenti di sviluppo, disegno tecnico e artistico, programmi di elaborazione di immagini, programmi di comunicazione e via dicendo. I programmi applicativi utilizzano i servizi offerti dal sistema operativo e da altre applicazioni. L'attività di richiesta effettuata da un software applicativo ad altri programmi o al sistema operativo viene

chiamata API ossia Application Program Interface che vuol dire interfaccia del programma applicativo.

Traduttori e linguaggi

Agli inizi degli anni '50 i programmatori lavoravano con i calcolatori della prima generazione. Essi venivano programmati con un linguaggio fatto solo di "0" ed "1" chiamato *linguaggio macchina* (machine language). Pensiamo ad uno schema del calcolatore nel quale vi è l'unità di controllo (U.C.) che invia segnali di controllo ai singoli elementi della macchina, ed in particolare che invia gli ordini di operazioni elementari ai singoli componenti della macchina. Il programma era costituito (ma lo è ancora oggi) da una sequenza di istruzioni. Le istruzioni più importanti erano le istruzioni di trasferimento che potevano essere suddivise in istruzioni di trasferimento *da registro a registro* ed istruzioni di trasferimento *da registro a memoria* e viceversa.

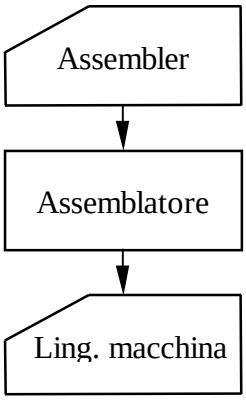


Supponiamo che la macchina avesse a disposizione un'istruzione in grado di ordinare il trasferimento del contenuto dell'accumulatore (ACC) nel registro B. Per ordinare questo trasferimento il programmatore scriveva una sequenza di cifre binarie più o meno suddivisibili idealmente in tre campi: nel primo campo scriveva ad esempio 000101 che costituiva il *codice operativo*. Tutte le volte che la macchina leggeva tale sequenza di cifre binarie in testa ad una istruzione capiva che quella è una istruzione elementare di tipo trasferimento da registro a registro. Poi ad esempio c'era scritto 000, che indicava il registro sorgente, il codice del registro ACC, 001 indicava il codice del registro di destinazione cioè il registro B.

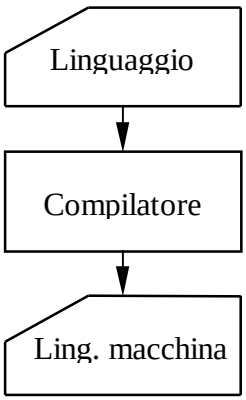
000101	000	001
Cod.Op.	ACC	B

Quindi il codice operativo indica il tipo di operazione ossia trasferimento registro-registro, mentre gli altri campi servono a specificare il registro sorgente e quello di destinazione. Nel caso di trasferimento registro-memoria o viceversa cambia il codice operativo e gli *indirizzi* di memoria. C'erano anche istruzioni di tipo **da registro a stampante**. I programmatori dovevano ricordare i codici operativi di tutte le istruzioni o eventualmente dovevano sfogliare il *manuale*, dovevano ricordare i codici dei registri, dovevano ricordare gli indirizzi delle variabili usate, cioè le posizioni di tutte le variabili in memoria. Il lavoro era lungo e difficile, il numero di errori era altissimo (basti pensare che su tantissime cifre binarie è facile invertire uno 0 con un 1). A tal fine era necessaria una fase di *debugging* ossia di eliminazione degli errori molto lunga.

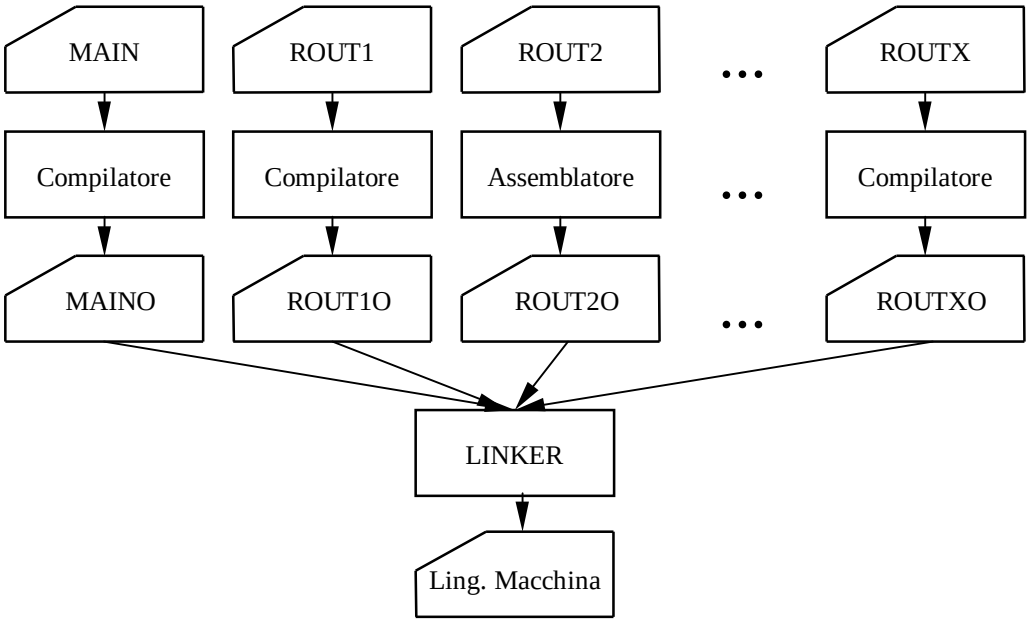
Per ovviare a questo tipo di difficoltà nacquero a metà degli anni '50 i primi *assemblatori* (in inglese assembler). Un assemblatore è un programma che è in grado di utilizzare un linguaggio di programmazione più facile del linguaggio macchina. Vediamo uno schema: in pratica il programma viene scritto in assembler e viene sottoposto all'assemblatore che lo traduce in linguaggio macchina.



L'idea era quella di lavorare con un linguaggio molto vicino a quello macchina ma i cui codici operativi non fossero delle sequenze (stringhe) di numeri ma parole più semplici da ricordare in quanto richiamanti le funzioni da eseguire. Poi vi era un programma che traduceva il tutto evitando la possibilità di errori di copiatura. Il linguaggio assembler ha una caratteristica fondamentale: una **corrispondenza uno a uno** tra le istruzioni del linguaggio assembler e quelle del linguaggio macchina. Ad esempio per operare una operazione di trasferimento fra registri adesso, molto più semplicemente, basta scrivere MOVE A, B; l'operatore non solo usa un *nome mnemonico* per il codice operativo ma usa anche delle **variabili** per rappresentare i registri, dato che è l'assemblatore che poi penserà a tradurle in indirizzi fisici svincolando quindi l'operatore dal ricordare gli indirizzi fisici della memoria dove aveva depositato dati e numeri. Ad esempio per depositare un valore dal registro A ad una cella di memoria bastava scrivere STORE A, PIPPO e per caricarlo sul registro B bastava digitare LOAD B, PIPPO. Gli assemblatori non sono molto complessi da realizzare, data la corrispondenza biunivoca fra linguaggio assembler e linguaggio macchina, e la loro diffusione fu un notevole passo in avanti. Ancora oggi quando si vogliono realizzare codici molto *efficienti* (molto veloci e che occupino poco spazio in memoria) si usano gli assemblatori.
Ma ciò non era sufficiente. Per esempio, i matematici desideravano poter scrivere espressioni aritmetiche molto più complesse come le formule, ad esempio A = (PIPPO * 2) / 3.1415, dove compaiono variabili, costanti e operatori. Tali linguaggi che nacquero dopo qualche tempo si chiamano *linguaggi di programmazione ad alto livello* per i quali viene persa la corrispondenza 1 a 1 a favore di funzionalità più ricche e di una maggiore facilità mnemonica essendo più vicini alla *logica umana* che alla macchina. Chiaramente i programmi scritti con tali linguaggi devono sempre essere tradotti in linguaggio macchina per poter essere interpretati dalla macchina. L'analogo dell'assemblatore per il linguaggio ad alto livello è il *compilatore*.



Lo schema logico è lo stesso: si parte da un programma scritto in linguaggio ad alto livello, lo si fa processare al compilatore (l'equivalente dell'assemblatore) che emette il programma scritto in linguaggio macchina.
Si noti che sia ora che precedentemente abbiamo usato nei diagrammi due simboli: un rettangolo e un rettangolo con uno spigolo tagliato. Questi ultimi vengono utilizzati per indicare una sorgente o una destinazione ossia qualcosa che deve essere elaborata o qualcosa che è stata elaborata. Questo simbolo ricorda che una volta i programmi erano costituiti da schede, mentre il simbolo del rettangolo pieno indica il programma compilatore che nel momento in cui è operante, è residente in memoria. Nello schema che abbiamo rappresentato per la compilazione c'è qualcosa di fondamentale che rimane sottinteso, qualcosa che rappresenta lo strumento principale attraverso cui la teoria del SW ha avuto uno sviluppo e un progresso elevatissimo: il **sottoprogramma**.



Chi scrive un programma scrive una parte principale chiamata *main* e utilizza tanti programmi già fatti da altri o da lui stesso. Da stime effettuate, di norma, solo il 3% del codice viene scritto ex novo, la parte rimanente sono *routine* e funzioni generali già scritte adattabili alle esigenze particolari. Il processo di traduzione diventa più complesso e viene effettuato in momenti diversi: il MAIN viene tradotto in un certo momento ottenendo un "codice oggetto" corrispondente che chiamiamo in figura MAINO (*oggetto*); analogamente si fa lo stesso per ROUT1, ROUT2, …, ROUTX (dove ROUT indica routine ossia sottoprogramma).
I rettangoli centrali possono essere traduttori di linguaggi ad alto livello cioè compilatori, oppure assemblatori (traduttori dal linguaggio assembler). I linguaggi ad alto livello potrebbero essere diversi per cui può accadere che il compilatore del linguaggio ad alto livello n.1 sia diverso dal compilatore del linguaggio ad alto livello n.2.
Per formare il programma in linguaggio macchina lavoro in due fasi successive: traduco separatamente, nella prima fase, tutti i *programmi sorgente* producendo dei moduli oggetto separati che poi utilizzo come strumento di ingresso per il programma concatenatore o **linker** che finalmente produrrà il linguaggio macchina.
Perché si è ricorsi ad uno schema così complicato?
Una prima difficoltà nasce dal fatto che in qualche caso il main viene scritto in un certo linguaggio mentre una o più routine sono scritte con linguaggi diversi o addirittura in assembler. Ma la ragione fondamentale che ha suggerito questo tipo di soluzione, è di ordine economico. Il processo di traduzione assorbe risorse significative dell'unità di calcolo (oggi percentualmente un po' meno dato il costo relativamente basso dei calcolatori): se scrivo un main coprendo al più il 3% del codice che

mi serve ed utilizzo, usando le *librerie* mi conviene utilizzare soltanto i moduli oggetto evitando di dover tradurre ogni volta routine che considero standard. Tali librerie infatti mettono a disposizione il codice macchina delle funzioni standard. Appunto per questo con i linguaggi di alto livello vengono fornite delle *librerie di routine* in formato oggetto. Così facendo si può risparmiare fino al 90% del lavoro di compilazione. Quindi parto dal mio programma sorgente che traduco e poi linko i moduli oggetto che mi tengo nella mia libreria ed il modulo oggetto che ho prodotto con il compilatore del linguaggio di alto livello. Utilizzando un particolare linguaggio, come il C, dovrò disporre del relativo compilatore che include le librerie e il linker. Normalmente le fasi di compilazione e di linkaggio non vengono distinte ossia vengono effettuate o meglio richiamate dallo stesso programma che sta compilando.