

UNIVERSITÀ MEDITERRANEA DI REGGIO  
CALABRIA

---

DiGiES

RACCOLTA DI MATERIALE DIDATTICO

**RACCOLTA DI MATERIALE DIDATTICO PER  
IL CORSO DI METODI INFORMATICI PER  
L'ECONOMIA**

MELCHIORRE MONACA

---

# Dalla matematica al Calcolatore Elettronico

## 1.1 La macchina di Turing

Nel 1936 Alan Turing pubblica l'articolo *On Computable Numbers with an application to the Entscheidungsproblem* [1], in cui viene descritta una macchina astratta, che altro non sarà che il principio di funzionamento di tutti i calcolatori a venire, destinata a diventare famosa come *Turing Machine*.

L'invenzione è tanto più ammirevole se si tiene conto che nel 1936 i computer non esistevano; solo alcuni anni più tardi, durante la guerra, Turing si trovò coinvolto nella costruzione di una macchina denominata *Colossus*, destinata alla decodifica dei messaggi criptati tedeschi.

Essendo ancora oggi il fondamento della scienza informatica, la Macchina di Turing è stata descritta in innumerevoli testi, in modi più o meno rigorosi e con un proliferare di varianti che, pur risultando tra loro equivalenti, si sono allontanate dalla formulazione originale. Qui si utilizzerà una “rappresentazione pittorica”, con l'unico scopo di fornire al lettore un'idea - semplificata, ma sostanzialmente corretta - della *Turing Machine*.

La *Turing Machine*, (che d'ora in poi abbrevieremo con *TM*), è un sistema formale, in grado di realizzare qualunque procedimento risolutivo di

un problema. È quello che in informatica (e in logica) si chiama comunemente *algoritmo* e, rispetto alla concezione comune di *risolvere un problema*, richiede che il ragionamento da effettuare sia discretizzato in passi, ovvero scomposto in una sequenza, per quanto lunga, di passaggi intermedi definiti, ciascuno descrivibile in modo chiaro e senza ambiguità.

La *TM*, è caratterizzata da un numero finito di condizioni, oggi abitualmente denominate “stati”, nell’originale “*m-configurations*” e da un “nastro” suddiviso in elementi, “squares” nell’originale, dove possono essere letti o scritti dei simboli.

Nell’articolo del 1936 si parlava esplicitamente di *computing machines* quando i simboli erano di due tipi: “0” e “1”: solamente i simboli binari erano suscettibili di interpretazione come dati. Era questa la versione in grado di computare numeri, rappresentati come cifre binarie. Più tardi, i simboli saranno considerati elementi di un insieme finito, che si chiama alfabeto.

La *TM* è composta da due parti:

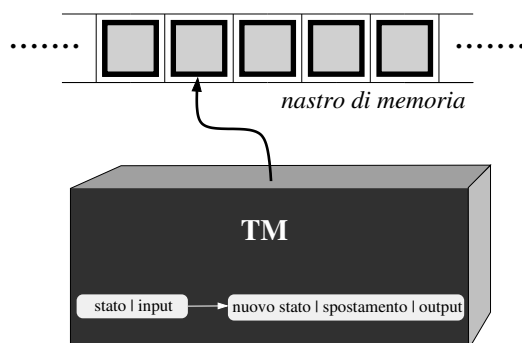
1. un’unità di memoria, esemplificata con una successione di caselle lungo un nastro.
2. un’unità di processo, che può assumere una serie finita di diversi stati

Vi è una casella della memoria che viene definita come casella di inizio, successivamente la *TM* potrà trovarsi in un’altra casella precedente o successiva a questa. Il passaggio da uno stato all’altro nell’unità di processo è regolato da una *tabella di istruzioni*, in cui lo stato attuale e la lettura della casella di memoria corrente, stabiliscono quale istruzione eseguire.

In ogni momento la *TM* si trova in un preciso stato e posizionata in una determinata casella: la combinazione di queste due condizioni, detta semplicemente “configuration” nell’originale, è tutto ciò che serve a stabilire il passo successivo, che può consistere solamente nelle seguenti mosse:

- leggere il dato nella casella di memoria corrente
- scrivere un nuovo dato nella casella di memoria corrente
- spostarsi in avanti nel nastro

- spostarsi indietro nel nastro
- cambiare il suo stato interno

Figura 1.1: Schema della *Turing Machine*

Nella Fig. 1.1 è visibile un semplice schema della *TM*, con l'essenza del suo funzionamento: dalla coppia stato-input viene deterministicamente fissato il suo comportamento, che consiste nella terna “nuovo stato/spostamento/output”. Tutte e tre queste mosse possono essere opzionali: se non c'è un nuovo stato vuol dire che quello attuale è *finale*, di termine della computazione; la macchina può rimanere nella stessa casella e può non produrre output (Turing standardizza questo caso come scrivere lo stesso simbolo letto). Devono essere fissati convenzionalmente uno stato iniziale e una casella iniziale e il funzionamento complessivo della macchina va specificato in una tabella dove sono scritte le istruzioni, sotto forma di corrispondenze tra configurazioni e comportamenti.

$\Downarrow$		0/1		
		0/1	0/1	

S	I	NS	NC	O
0	-	1	$\Rightarrow$	0
1	-	0	$\Rightarrow$	1

Tabella 1.1: Esempio di una *TM* che produce la sequenza infinita 0101...

In Tabella 1.1 viene schematizzata una  $TM$  che produce la sequenza infinita 0101... In alto il nastro di memoria, qui rappresentato con la possibile scelta di valori (alfabeto dei simboli), in basso la tabella delle istruzioni. La freccia verticale sul nastro indica la casella d'inizio. Le colonne della tabella contengono, nell'ordine: **S**tato, **I**nput, **N**uovo **S**tato, **N**uova **C**asella, **O**utput.

Il funzionamento passo dopo passo di un'istanza di questa  $TM$  è rappresentato in Tabella 1.2.

↓				
	0			
S	I	NS	NC	O
0	-	1	⇒	0
1	-	0	⇒	1

↓				
	0	1		
S	I	NS	NC	O
0	-	1	⇒	0
1	-	0	⇒	1

↓				
	0	1	0	
S	I	NS	NC	O
0	-	1	⇒	0
1	-	0	⇒	1

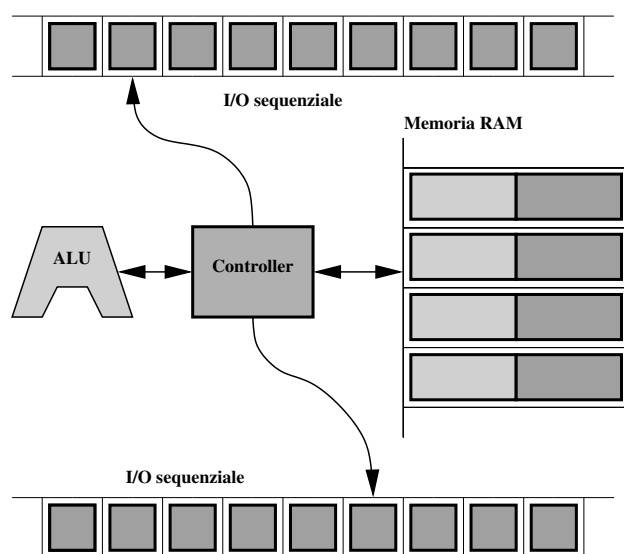
↓				
	0	1	0	1
S	I	NS	NC	O
0	-	1	⇒	0
1	-	0	⇒	1

Tabella 1.2: Esecuzione passo passo di una  $TM$  che produce la sequenza infinita 0101...

Il concetto di  $TM$  può essere esteso in vari modi rispetto a quanto qui descritto e formalizzato con rigore matematico, per una trattazione più approfondita si rimanda alla bibliografia citata.

## 1.2 La RAM machine

Proseguendo nel passaggio dalla macchina di Turing al computer reale, rimangono dei vuoti che l'astrazione dell'idea di  $TM$  aveva lasciato, in partico-

Figura 1.2: Modello della **RAM Machine** di von Neumann

lare la realizzabilità della memoria (difficile nel concetto di nastro sequenziale infinito) e il fatto che l'unica possibile mossa sia la transizione di stati.

Va a John von Neumann, matematico ungherese naturalizzato americano, il merito storico di aver derivato dal modello della *TM* un'altra macchina astratta, formalmente equivalente, ma direttamente utilizzabile come schema per progettare reali computer. Questa macchina è denominata talvolta *IAS architecture* dall'Institute of Advanced Studies di Princeton dove von Neumann lavorava, oppure *RAM machine*, per la sua caratteristica di avere una memoria indirizzabile direttamente, o infine semplicemente *von Neumann architecture* [2].

Le componenti di questo nuovo modello sono visibili nella figura 1.2. La novità principale è appunto la memoria cosiddetta RAM, da *Random Access Memory*, termine che non molto felicemente sta a indicare la possibilità di accedere direttamente ad un dato in memoria tramite il suo indirizzo. Random access (letteralmente “accesso casuale”) potrebbe far intendere che questo dispositivo è in grado di fornire casualmente uno dei suoi tanti dati memorizzati, così come un dado può fornire un numero casuale tra 1 e 6. Quello che invece succede, è come se nello specificare la nuova casella nella

$TM$ , anziché  $\Rightarrow/\Leftarrow/-$ , si potesse usare un numero, il cosiddetto indirizzo (*address*), che rappresenta la posizione della casella su cui la macchina si posizionerà successivamente. Il concetto di RAM permette la preziosa libertà di poter accedere ad una cella di memoria qualunque, con tempi che non dipendono da qual'è l'ultima casella visitata, ma perde qualcosa della generalità del nastro di memoria di Turing, in quanto adesso ogni casella deve avere rigidamente assegnato a priori un indirizzo, e chi scrive un programma deve conoscere quali sono gli indirizzi disponibili di una RAM.

L'altra novità importante è la componente ALU (*Arithmetic-Logical Unit*), che risponde all'esigenza di rendere disponibili operazioni di uso frequente, evitando quindi di doverle realizzare ogni volta mediante passaggi di stato. La ALU può essere pensata come una collezione di tante  $TM$  dedicate: una per l'addizione, una per la congiunzione logica, una per la disgiunzione e così via, ciascuna realizzata con un suo specifico circuito elettronico. Chi scrive il programma, non deve più occuparsi di come implementare queste operazioni, è sufficiente che dia alla ALU l'indicazione che deve attivare una particolare operazione. Anche questa innovazione è a scapito della massima generalità del sistema di Turing, che consentiva di implementare qualunque algoritmo basandosi sull'unico meccanismo del passaggio di stati: adesso per scrivere un programma è necessario conoscere quali sono le modalità che attivano in una ALU l'operazione desiderata.

Viene adottato allo scopo il concetto di *istruzione*, un codice univoco, memorizzabile come sequenza binaria, che la ALU riconosce come comando di attivare i circuiti che provvedono ad una certa operazione. L'insieme di tutte le istruzioni riconosciute dalla ALU va sotto il nome di *instruction set*. Non esiste un insieme di istruzioni standardizzato, ogni progettista di ALU sceglie le proprie, tuttavia esistono alcune categorie di massima di istruzioni, comuni alla maggior parte delle ALU:

- Trasferimento dati: istruzioni che servono a spostare dati da una posizione di memoria all'altra;

- Aritmetiche: istruzioni per operazioni, appunto, aritmetiche, quali addizione, sottrazione, moltiplicazione e talvolta divisione;
- Logica: istruzioni per eseguire congiunzione (AND), disgiunzione (OR) e altre operazioni logiche;
- Controllo: istruzioni che fanno passare l'esecuzione ad un altro punto del programma: realizzano l'equivalente delle transizioni di stato della *TM*.

La Ram Machine è totalmente programmabile: il programma da eseguire è codificato e memorizzato nella RAM, esattamente come gli altri dati. Le istruzioni che compongono un programma saranno pertanto sequenze binarie residenti in celle della memoria RAM, da cui vengono prelevate dall'unità di controllo e copiate nella ALU. L'unità di controllo inoltre scandisce le sequenze di operazioni del sistema. Continuano ad esistere altri dispositivi di memoria ad accesso sequenziale, che rappresentano i modi con cui il computer colloquia con il mondo esterno, ma l'esecuzione del programma avviene soltanto attraverso la memoria RAM.

## 1.3 Dalla RAM machine... la CPU

Il modello RAM machine ha avuto un tale successo da soppiantare ogni alternativa proposta agli albori dei computer e si è andato sempre più affermando come unica ricetta per costruirli, pienamente valida ancora oggi. La sua versione consolidata e applicabile a qualunque computer odierno è un'architettura semplice ed elegante, sintetizzabile nello schema di figura 1.3. La cosiddetta CPU (*Central Processing Unit*) dello schema corrisponde all'insieme della ALU e del Controller nel modello RAM Machine ed è quella parte del computer dedicata all'elaborazione dei dati, che avviene eseguendo istruzioni che facciano parte del suo instruction set.

La memoria corrisponde alla memoria RAM, che contiene sia il programma da eseguire, cioè la sequenza di istruzioni che risolvono il compito deside-



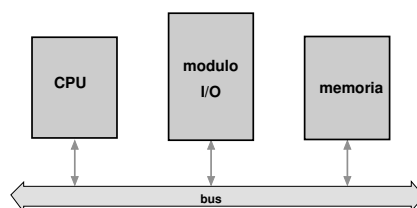


Figura 1.3: Componenti Hardware di un computer

rato, sia i dati necessari al programma, ed è l'unico tipo di memoria con cui la CPU scambia dati direttamente. Ogni altro componente del computer è visto nello schema come un modulo I/O (*input/output*), corrispondente alle memorie sequenziali della RAM machine.

In termini esemplificativi si può dire che la CPU ha trovato il suo miglior modo di esprimersi mediante l'accesso RAM, in cui può leggere o scrivere qualunque dato semplicemente tramite un numero che rappresenta il suo indirizzo, e pertanto non gradisce più colloquiare con componenti che parlano in modo sequenziale.

Rimane però l'esigenza di colloquiare ogni tanto anche con l'esterno, che a differenza della RAM è necessariamente ad accesso sequenziale: allo scopo si è introdotto questo elemento di mediazione, il modulo I/O, che si prende il compito di interagire con un dispositivo sequenziale, mascherandolo poi nei confronti della CPU come se si trattasse di un pezzo di RAM. Quindi, quando la CPU deve scambiare un dato che non è in RAM, continua imperterrita ad usare un numero come suo indirizzo, ma in questo caso il numero corrisponde, per il sistema, ad un modulo di I/O, che riceve la richiesta della CPU e provvede ad interagire con l'elemento sequenziale. Questa è la modalità generale con cui la CPU vede tastiere, mouse, schede audio, floppy drive, stampanti, hard disk, USB key, CD-ROM e così via: tanti moduli I/O, ciascuno con il proprio indirizzo.

Il mondo dei computer conosciuto oggi si apre di fatto con l'invenzione del microprocessore, introdotto dall'Intel nel 1971. Quest'elemento, chiamato anche CPU (*Central Processing Unit*) è diventato il fulcro attorno a cui si progetta il resto di un computer e che in buona parte ne determina le

prestazioni complessive.

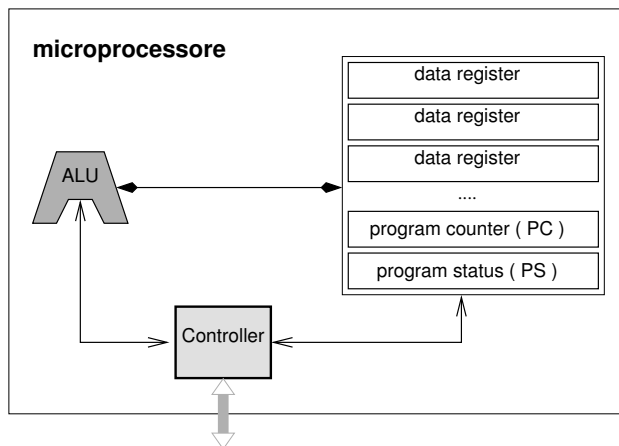


Figura 1.4: Componenti principali di un microprocessore

L'architettura di massima di un microprocessore è quella di Fig. 1.4, notare la somiglianza con la *RAM Machine* di von Neumann (in Fig. 1.2). La singolarità è che quello era lo schema dell'intero computer, mentre la CPU ne è solo un componente, assimilabile alla ALU della *RAM Machine*.

La somiglianza con la *RAM Machine* è immediatamente evidente per i componenti ALU e controller, l'equivalente della memoria RAM per il microprocessore è il suo set di registri. Come nella RM, anche qui, la ALU è piuttosto elitaria nella sua comunicazione: parla solo e soltanto con i suoi registri. Questi costituiscono quindi le aree di memoria da cui la ALU prende i dati necessari per le sue operazioni e su cui scrive i risultati. I registri sono in genere in numero limitato, insufficiente a mantenere tutti i dati necessari durante l'esecuzione di un programma. Pertanto essi dovranno essere scambiati con la memoria RAM, di questo si occupa il controller.

Con una semplificazione si può riassumere così l'operare della CPU: la ALU opera sui dati nei registri e ogni volta che occorre elaborare dati che non sono già disponibili, il controller li preleva dalla RAM esterna e li copia nei registri stessi.

Entriamo ora più all'interno della CPU, per vedere come avviene, in questo schema semplificato, il suo funzionamento. Il principio è esattamente

quello della RM: viene letta dalla RAM un'istruzione per volta, ed eseguite le operazioni codificate in tale istruzione. Tali operazioni possono avere come argomenti dei dati o dei numeri direttamente codificati nell'istruzione. Per i dati, il caso migliore è che questi si trovino già nei registri, l'operazione è allora eseguita immediatamente. Se non è così, il controller deve richiedere, specificandone l'indirizzo, il dato all'esterno e trasferirlo in un registro.

Come sapere qual'è l'indirizzo in memoria dove è immagazzinata l'istruzione da eseguire? Possiamo vedere nella figura 1.4, che mentre i primi registri sono disponibili per contenere qualunque dato su cui eseguire operazioni, gli ultimi hanno dei nomi diversi, che corrispondono a delle loro funzioni particolari, questi registri dedicati non sono scritti dall'utente, ma direttamente dalla circuiteria interna della CPU. Il registro denominato PC (*Program Counter*) ha il compito di contenere l'indirizzo in memoria dell'istruzione da andare a prelevare ed eseguire.

Il funzionamento del *Program Counter* è banale, si basa sul presupposto che un programma sia residente nella memoria con una sequenza ordinata, con ogni istruzione seguita da quella che è opportuno venga eseguita successivamente. Ciò pare una regola di normale buonsenso, ma si sottolinea come non è affatto scontato debba essere così, le istruzioni della macchina di Turing sono scritte nella tabella in un ordine convenzionale, del tutto indipendente dall'ordine in cui sono eseguite. La CPU si adegua al senso comune, pertanto l'esecuzione di un'istruzione incrementa il *Program Counter* semplicemente all'indirizzo di memoria successivo. A meno che l'istruzione in corso non appartenga alla categoria di quelle di controllo: in questo caso l'istruzione stessa conterrà l'indicazione del nuovo indirizzo in memoria, che verrà impostato nel *Program Counter*.

Per chiarire quanto detto si inizia a mostrare il comportamento della CPU nell'esecuzione di un frammento elementare di programma (scritto in un particolare linguaggio, l'*assembler*), schematizzato in Fig. 1.5. Prima dell'esecuzione tutti i registri della CPU hanno valori arbitrari, eccetto il PC, che contiene 1000, indirizzo della RAM con la prima istruzione. La numerazione

della RAM nella figura potrà apparire un pò strana, con il 1009 seguito dal 100A: è un dettaglio minore. La notazione in uso è quella esadecimale, cioè in base 16, basata sulle cifre da 0 a 9 seguite dalle lettere dalla A alla F (16 possibili valori per ogni cifra).

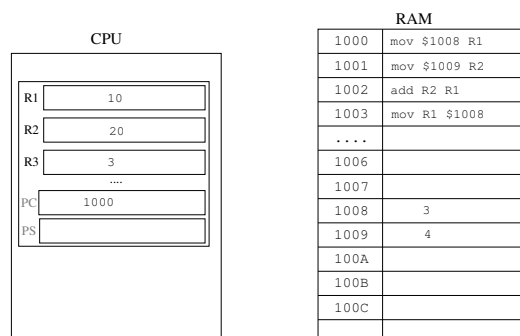


Figura 1.5: Esempio di un frammento di assembler, usato per mostrare il funzionamento di una CPU.

Tornando al programma di Fig. 1.5, altre piccole precisazioni sui contenuti della RAM. Le istruzioni, così come i dati, sono sequenze di bit, per renderle comprensibili nella figura sono state scritte nel linguaggio *assembler*, una convenzione linguistica per assegnare nomi alle istruzioni e specificare gli operandi. Per la precisione non esiste un unico assembler, ve ne sono diversi, legati alle varie CPU, qui non se ne usa nessuno specifico, ma una sua forma tipica.

L'istruzione di spostamento si chiama `mov`, ed è seguita da due operandi: il dato da prendere e dove metterlo.

I registri vengono chiamati direttamente `R1`, `R2`, mentre le locazioni in memoria RAM richiedono qualche accorgimento sintattico, il simbolo `$` davanti ad un numero. Così si capisce che non va usato il numero stesso, ma che quel numero è un indirizzo e il dato è il contenuto della cella di RAM corrispondente a quell'indirizzo.

Nella Fig. 1.6 è mostrato il dettaglio dell'esecuzione della prima istruzione. Il registro PC vale 1000, ciò vuol dire che l'istruzione da eseguire si trova in RAM ad indirizzo 1000.

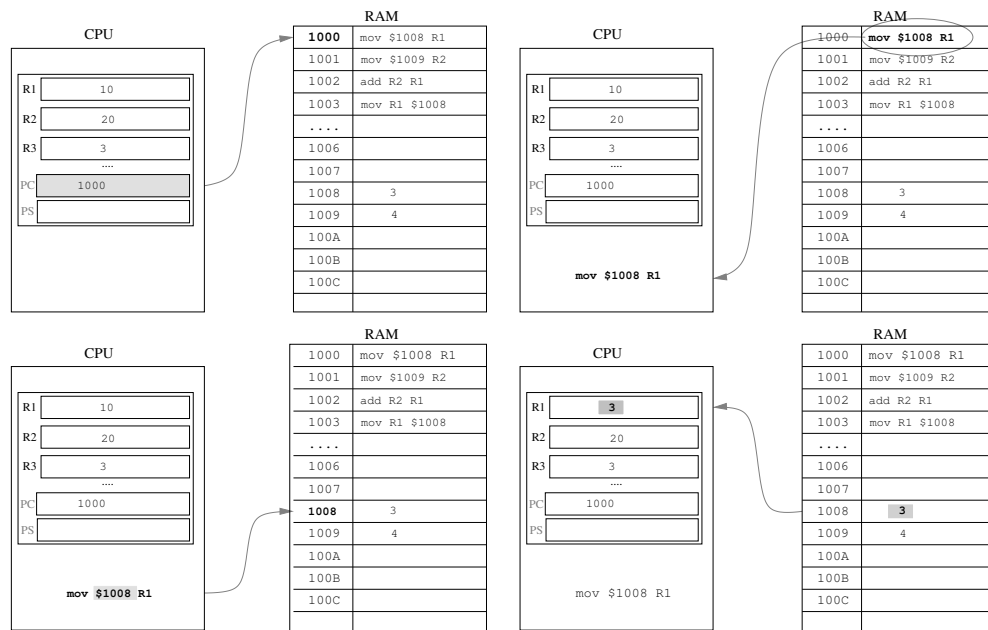


Figura 1.6: Esempio di esecuzione della prima istruzione dell'assembler in RAM (da sinistra a destra, dall'alto in basso).

Quest'istruzione:

```
mov $1008 R1
```

viene caricata dalla RAM all'interno della CPU, per essere eseguita. In termini discorsivi, l'istruzione dice di prendere il dato all'indirizzo 1008 della RAM e metterlo nel registro R1. Quindi, (Fig. 1.6 in basso a sinistra), la CPU deve fare un accesso alla RAM all'indirizzo 1008, leggendo il dato nel registro R1, che varrà ora 3 (in basso a destra).

Il resto del programma è mostrato (in modo più sintetico), nella Fig. 1.7: in **a** si vede l'avanzamento del PC, che da 1000 è passato a 1001 e quindi l'istruzione caricata nella CPU è

```
mov $1009 R2
```

che come conseguenza scrive il valore 4 in R2. In **b**, con il PC a 1002, viene caricata la prima istruzione della categoria aritmetica:

```
add R2 R1
```

anche questa istruzione ha due operandi, che devono essere registri e il secondo operando è anche la destinazione del risultato; come si vede in Fig. 1.7 c, il registro R1 diventa 7. Infine, nei riquadri **d** e **e**, è raffigurata un'altra istruzione di spostamento, che questa volta mette il risultato dell'addizione in RAM, alla locazione 1008.

Come accennato, non sempre il PC si comporta nel modo elementare appena visto. Vi sono in generale due tipi di istruzioni che interrompono il flusso normale del programma: quelle *incondizionate* e quelle *condizionate*.

Le prime si spiegano da sole: eseguono sempre il *salto* ad un punto diverso del programma. Le seconde invece possono sia provocare questo salto, oppure mantenere la sequenza normale, a seconda di una condizione.

È il tipico caso di istruzioni che corrispondono alla *if* dei linguaggi ad alto livello. Il problema è che la condizione è di solito il risultato di un'operazione e quindi a sua volta di un'istruzione che non è quella di controllo di cui stiamo parlando. Per esempio, se la *if* contiene la condizione "se un dato è maggiore di un certo numero", questo comporta eseguire la differenza del dato e del numero e vedere se il risultato è maggiore, minore o uguale a 0. È necessario che la CPU abbia quindi eseguito prima l'istruzione corrispondente alla sottrazione, ma che il suo risultato sia in qualche modo conservato per capire come deve comportarsi l'istruzione di salto condizionale successiva. A questo provvede un altro registro dedicato, il PS (*Program Status*), che è sempre scritto dalla CPU, ma è leggibile dall'utente come fosse un normale registro.

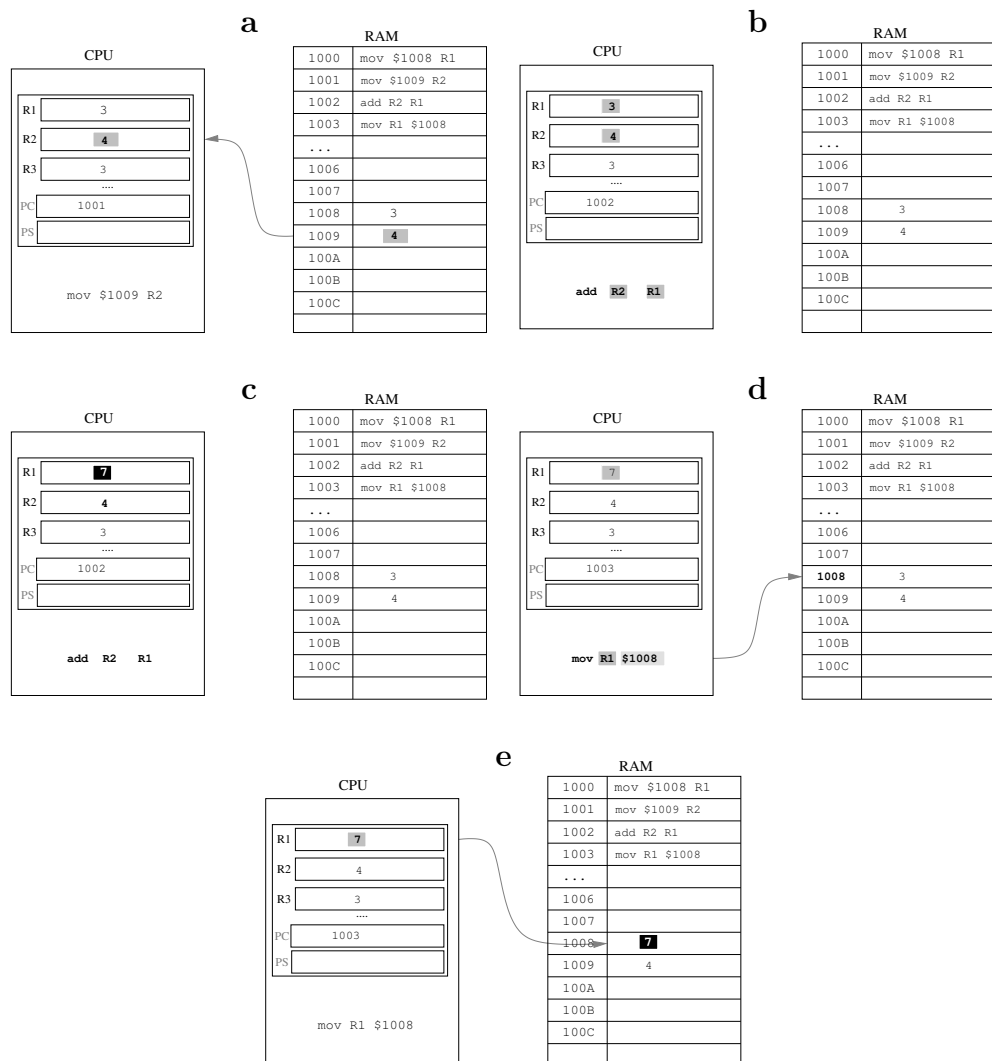


Figura 1.7: Esempio di esecuzione dell'assembler in RAM, oltre all'istruzione illustrata in dettaglio nella 1.6.

# Capitolo 2

## Elementi di basi di dati

Esiste una certa varietà di architetture atte a svolgere il ruolo di base dati, e analogamente diversi linguaggi per il loro trattamento. In questa presentazione a carattere introduttivo, ad una rassegna superficiale si è preferito entrare nel merito di un solo tipo di data base e di un solo linguaggio. Questo anche perché entrambi sono quelli di gran lunga oggi dominanti e adatti all'impiego in ambito web. Per una rassegna più ampia si veda, tra i tanti, [3].

### 2.1 Insiemi e relazioni

Il termine *relazionale* è stato introdotto in [4], ed ha caratterizzato un modello di database che progressivamente è andato affermandosi, superando quelli già esistenti, diventando dagli anni '80 in poi quello più popolare. Deve il suo nome al concetto di *relazione* in teoria degli insiemi. Più precisamente in questo contesto la relazione è un sottoinsieme del prodotto cartesiano su *domini*, che altro non sono che insiemi i cui elementi sono omogenei. Per esempio sono tutti numeri interi, oppure stringhe non più lunghe di un dato numero di caratteri, giorni della settimana, e così via.

Per esempio una relazione  $S$  che contenga informazioni su delle persone



può essere data da:

$$S \subseteq N \times C \times A$$

dove  $N$  è il dominio dei nomi,  $C$  dei cognomi e  $A$  contiene anni. Pertanto  $S$  avrebbe elementi del tipo:

$$S = \{\langle \text{Elisa, Rossi, 2001} \rangle, \langle \text{Carlo, Bianchi, 1947} \rangle, \dots\}$$

Gli elementi di una relazione in generale sono  $n$ -ple, in particolare quelli di  $S$  sono triple. Un sinonimo di relazione molto in uso nei database è quello di tabelle (*tables*), appropriato perché i domini possono essere visti concettualmente come colonne di una tabella di dati, e così infatti sono denominati; anche se in questo gergo pratico viene meno il riferimento al fondamento matematico del modello.

Un database quindi non è altro che un insieme di relazioni, atte a contenere le informazioni desiderate. Tipicamente ci saranno domini in comune tra diverse relazioni. Nell'esempio di sopra, si può immaginare una relazione  $L$  del genere:

$$L \subseteq A \times P$$

dove  $A$  sono gli anni, esattamente come in  $S$ , e  $P$  è un numero che rappresenta la popolazione totale in quell'anno, gli elementi quindi saranno:

$$L = \{\langle 1947, 20000000 \rangle, \langle 2001, 60000000 \rangle, \dots\}$$

Uno dei domini costituenti una relazione può essere una sua *chiave* (*key*), la proprietà è di essere unica, non ci possono essere due elementi della relazione in cui quel costituente è identico. Nel caso delle persone si potrebbe introdurre una chiave ampliando la tabella nel modo seguente:

$$S \subseteq I \times N \times C \times A$$

dove  $I$  potrebbe essere il codice della carta di identità della persona, oppure semplicemente un numero progressivo mantenuto convenzionalmente nella tabella.

## 2.2 La logica del selezionare

Esiste un linguaggio dedicato ai database relazionali, la cui essenza è la capacità di esprimere criteri logici con cui selezionare dall'archivio l'informazione desiderata. È stato sviluppato dalla IBM a metà degli anni '70, dapprima con la denominazione di SEQUEL (acronimo di *Structured English Query Language* [5], attualmente SQL (*Standard Query Language*). Standard a pieno titolo, non solo per la sua adozione all'interno di ANSI/ISO, ma per essere divenuto dagli anni '80 in poi il linguaggio per eccellenza nel dialogare con database relazionali. Una delle implementazioni attuali più diffusa è il MySQL, disponibile come open source [6].

Il termine *Query* nell'acronimo è distintivo di questo genere di linguaggi, perché la funzione centrale di un sistema di database è proprio la capacità di selezionare, all'interno dell'archivio di dati, tutti e solo quelli di volta in volta desiderati. Il cuore del linguaggio è pertanto la sua capacità di formulare “interrogazioni” verso le relazioni del database in modo potente ed efficace, e contenere all'interno le modalità matematiche per realizzarle. La query è quindi la componente principale, ma non l'unica in un linguaggio come SQL, occorre disporre anche di modi per creare le relazioni e introdurre dati. Sono operazioni concettualmente molto meno complesse, ma verranno qui introdotte prima per rispettare un'ipotetica sequenza temporale nel creare e poi interrogare un database.

Le relazioni che interessano devono essere definite all'interno di uno specifico database, pertanto se non ne esiste già uno, occorre anzitutto crearlo, con il comando:

```
CREATE DATABASE saperi;
```

dove **saperi** è il nome che si vuol dare al nuovo database. Per facilitare il lettore si sta usando la convenzione di scrivere in maiuscolo i termini che fanno parte del linguaggio SQL, e in minuscolo tutti gli altri. Da notare il punto e virgola, particolarità sintattica obbligatoria per chiudere qualunque coman-

do SQL. Per indicare che tutte le successive operazioni sono da intendersi all'interno del database `saperi` basta scrivere:

```
USE saperi;
```

A questo punto è possibile cominciare a creare relazioni, anche queste avranno un loro nome, del tutto arbitrario, e richiederanno la definizione del tipo di elemento contenuto in ciascun dominio. Ecco il comando per creare la relazione dell'esempio di sopra:

```
CREATE TABLE persone (  
    id            INT      NOT NULL AUTO_INCREMENT,  
    nome          TEXT     NOT NULL,  
    cognome       TEXT     NOT NULL,  
    anno          YEAR,  
    cosa_fa       INT,  
    PRIMARY KEY  ( id )  
);
```

Dopo l'intestazione del comando, `CREATE TABLE persone`, segue tra parentesi tonda la lista dei domini, ciascuno con un proprio nome e una dichiarazione di tipo. Esiste una varietà di tipi accettati, che sostanzialmente si dividono in dati numerici o stringhe. Per `id` si è usato un tipo numerico, mentre `cognome` e `nome` sono stringhe di caratteri. Nell'ultima riga con `PRIMARY KEY` viene designato il dominio che funge da chiave, in questo caso `id`, che per poter svolgere il ruolo di chiave deve avere certe prerogative. In SQL è consentito che elementi della relazione abbiano qualche termine mancante, che equivale a ritenere presente nei vari domini un termine convenzionalmente considerato nullo. Non però se si tratta del dominio preso come chiave, e la limitazione è esplicitata dalla dichiarazione `NOT NULL`, nella definizione di `id`. L'ulteriore indicazione `AUTO_INCREMENT` per il dominio `id` sarà utile durante l'operazione di inserimento, come si vedrà immediatamente a seguito.

Anche per `nome` e `cognome` si è adottato `NOT NULL`, in quanto ha poco senso l'esistenza di un record riguardante una persona di cui non si conosce

nemmeno nome e cognome. È plausibile che invece non si sappia il suo anno di nascita, e non si abbiano neppure le idee chiare su cosa abbia fatto nella vita per essere ricordata questa persona. Pertanto sia `anno` che `cosa_fa` mancano della specifica `NOT NULL`. Notare che aver scritto l'istruzione `CREATE` su più righe, e con quell'allineamento, è un fatto puramente estetico, volto a evidenziare la struttura dell'istruzione per chi la deve leggere e capire. Per SQL non c'è invece nessuna regola in tal senso, si può a piacere scrivere tutto su un'unica riga o ripartirla e formattarla come si vuole: la conclusione dell'istruzione è segnalata dalla presenza del punto e virgola.

Potrebbe sembrare singolare che il dominio `cosa_fa` sia di tipo numerico, dire cosa qualcuno faccia nella vita dovrebbe richiedere una descrizione testuale. Il punto è che tale descrizione rischierebbe di non essere standardizzata, rendendo il sistema poco adatto a cercare in modo efficiente i propri contenuti. Per esempio una persona potrebbe essere classificata come “pittoré”, un altro come “artista pittorico”. Pur facendo la stessa cosa, in una ricerca basata sul campo `cosa_fa` non figurerebbero insieme. Ciò che si fa in questi casi è usare un codice, e una seconda relazione, che servirà a raggruppare e descrivere opportunamente tutti i possibili codici, in questo caso le possibili attività delle persone archiviate nel database:

```
CREATE TABLE cosa_si_fa (  
    id          INT      NOT NULL AUTO_INCREMENT,  
    denom       TEXT     NOT NULL,  
    descr       TEXT,  
    PRIMARY KEY ( id )  
);
```

dove `denom` sarà una denominazione breve di “cosa fanno” le persone, mentre `descr` una descrizione più estensiva.

Finora si sono costruite solamente le strutture in cui possono trovare alloggio le informazioni, ora può iniziare la fase della loro immissione. Per questo scopo c'è l'istruzione `INSERT INTO`, di cui si può vedere la sintassi in questi esempi in cui viene popolata la relazione `persone`:

```
INSERT INTO persone
    VALUES( 1, 'Alan', 'Turing', 1912, 1 );
INSERT INTO persone
    ( nome, cognome, cosa_fa, anno )
    VALUES( 'Claude', 'Shannon', 1, 1916 );
INSERT INTO persone
    ( cognome, nome, cosa_fa )
    VALUES( 'Clausius', 'Rudolf', 2 );
INSERT INTO persone
    SET nome='Jared', cognome='Diamond';
```

dove si vede che immediatamente dopo INTO deve seguire il nome della relazione, poi vi sono diverse alternative per fornire i valori. Nel primo caso si è scritto direttamente VALUES, termine che introduce, entro parentesi, la lista delle informazioni da inserire. In questa forma semplice sono necessarie due accortezze: la lista deve contenere tutti i campi definiti nella tabella, e il loro ordine deve seguire rigorosamente quello usato in fase di creazione della tabella con il comando CREATE TABLE, in questo caso id, nome, cognome, cosa\_fa, anno. Nel secondo formato, VALUES viene preceduto da un'altra lista, sempre tra parentesi tonde, con l'elenco dei campi che si intende scrivere. È infatti possibile omettere in un elemento della relazione i termini di qualche dominio, per esempio anno nel caso di Rudolf Clausius. Come già detto ciò non può valere per il dominio chiave, in questo caso id, ma in nessuno degli esempi sopra viene incluso tra i domini validati. Ciò è possibile grazie alla proprietà AUTO\_INCREMENT, che indica al database di inserirlo automaticamente qualora sia stato omissso, incrementando semplicemente di 1 rispetto all'ultimo valore presente nella relazione. Notare l'uso degli apici ' per delimitare le stringhe. Il terzo tipo di sintassi di INSERT INTO ha una forma semplificata: si è omissa la lista con i nomi dei domini, al posto di VALUES si usa il termine SET, in cui si validano i domini desiderati mediante =.

Ed ecco un paio di inserimenti nella relazione cosa\_si\_fa.

```
INSERT INTO cosa_si_fa
```

```
VALUES ( 1, 'matematico', 'si dedica allo studio
dei numeri, della logica, e cose del genere' );
INSERT INTO cosa_si_fa
SET denom='fisico';
```

Il comando adibito alle query è **SELECT**, che presenta una varietà di sintassi, qui introdotte partendo dalle più semplici. Il comando:

```
SELECT id, cognome FROM persone;
```

produce la tabella:

```
+----+-----+
| id | cognome |
+----+-----+
|  1 | Turing  |
|  2 | Shannon |
|  3 | Clausius |
|  4 | Diamond |
+----+-----+
```

da cui si evince che dalla relazione **persone**, specificata dopo **FROM**, sono stati prelevati tutti gli elementi, e al loro interno estratti solamente i valori corrispondenti ai domini selezionati dopo **SELECT**, ovvero **id**, **cognome**. Una forma ancor più semplificata è la:

```
SELECT * FROM persone;
```

che produce la tabella:

```
+----+-----+-----+-----+-----+
| id | nome   | cognome | anno | cosa_fa |
+----+-----+-----+-----+-----+
|  1 | Alan   | Turing  | 1912 | 1 |
|  2 | Claude | Shannon | 1916 | 1 |
|  3 | Rudolf | Clausius | NULL | 2 |
|  4 | Jared  | Diamond | NULL | NULL |
+----+-----+-----+-----+-----+
```

Il simbolo \*, uno dei cosiddetti *wildcard characters* è l'abbreviazione per tutti i domini della relazione. Notare nel risultato come per i due persone di cui non è nota l'età venga visualizzato `NULL`, indicatore di valore omesso.

L'ordine in cui vengono presentati gli elementi è quello del loro inserimento, ma è possibile specificare un ordinamento desiderato, per esempio:

```
SELECT id, cognome FROM persone ORDER BY cognome;
```

con effetto:

```
+----+-----+
| id | cognome |
+----+-----+
|  3 | Clausius |
|  4 | Diamond  |
|  2 | Shannon  |
|  1 | Turing   |
+----+-----+
```

Fin qui l'uso di `SELECT` non ha reso giustizia al suo nome, nel senso che sono stati presentati tutti gli elementi della relazione, la potenza del comando si esplica invece soprattutto quando uno vuole, appunto, selezionare solamente alcuni elementi di interesse. Ciò avviene completando `SELECT` con la clausola `WHERE`, seguita da una espressione logica, ovvero che restituisca valori vero/falso, eventualmente composta tramite i connettivi logici `NOT`, `AND` a `OR`. Un primo esempio:

```
SELECT id, cognome FROM persone
      WHERE cosa_fa=1;
```

```
+----+-----+
| id | cognome |
+----+-----+
|  1 | Turing  |
|  2 | Shannon |
+----+-----+
```

L'espressione logica `cosa_fa=1` restituisce `vero` nel caso in cui l'identificativo dell'attività della persona sia 1 e `falso` in tutti gli altri casi, pertanto sono stati selezionati solo i matematici. Esiste un operatore di uso frequente per la comparazione di stringhe, che restituisce valori logici: `LIKE`, eccone un esempio di uso:

```
SELECT nome, cognome FROM persone
      WHERE cognome LIKE '%u%';
```

l'espressione che segue `LIKE` contiene un altro *wildcard character*, `%`, che rappresenta una sequenza generica di qualunque carattere, per cui `LIKE '%u%'` restituisce `vero` per tutte le stringhe che contengano `u`. Il risultato è:

```
+-----+-----+
| nome  | cognome |
+-----+-----+
| Alan  | Turing  |
| Rudolf | Clausius |
+-----+-----+
```

Ed ecco come comporre la clausola `WHERE` se si volessero selezionare i matematici il cui nome inizi per `A`:

```
SELECT id, cognome FROM persone
      WHERE cosa_fa=1 AND nome LIKE 'A%';
```

```
+----+-----+
| id | cognome |
+----+-----+
|  1 | Turing  |
+----+-----+
```

La `SELECT` consente non solo di selezionare elementi di una singola relazione, ma di costruire nuovi elementi dal prodotto cartesiano di più relazioni. L'effetto è subito illustrabile con un esempio: si supponga di voler elencare le persone per cosa fanno, espresso non in codice ma con la sua denominazione



completa, disponibile dalla relazione `cosa_si_fa`. Per costruire il prodotto cartesiano di più relazioni basta far seguire alla clausola `FROM`, anziché il nome di una singola relazione, un elenco con quelle desiderate:

```
SELECT persone . cognome, cosa_si_fa . denom
      FROM persone, cosa_si_fa;
```

produce la tabella:

cognome	denom
Turing	matematico
Turing	fisico
Shannon	matematico
Shannon	fisico
Clausius	matematico
Clausius	fisico
Diamond	matematico
Diamond	fisico

Notare che adesso per designare i domini occorre usare una sintassi del tipo  $\langle R \rangle . \langle D \rangle$ , nome della relazione e separato da un punto nome del dominio. Il risultato ottenuto avrà probabilmente scontentato il lettore attento: i cognomi delle persone sono ripetuti due volte, e senza nessuna correlazione con la loro effettiva attività. Ed è esattamente ciò che ci si deve attendere dall'uso fatto di `SELECT`, perché come detto matematicamente non fa che costruire una nuova relazione:

$$R' = S \times L$$

dove  $S$  è la relazione `persone` e  $L$  `cosa_si_fa`. Per ottenere invece il risultato desiderato è sufficiente selezionare un sottoinsieme di  $R'$  che soddisfi alla proprietà che il codice numerico dell'attività sia lo stesso per il personaggio e per la denominazione dell'attività:

```
SELECT persone . cognome, cosa_si_fa . denom
      FROM persone, cosa_si_fa
      WHERE persone . cosa_fa = cosa_si_fa . id;
```

produce la tabella:

```
+-----+-----+
| cognome | denom   |
+-----+-----+
| Turing  | matematico |
| Shannon | matematico |
| Clausius | fisico    |
+-----+-----+
```

che stavolta corrisponde a ciò che si intendeva chiedere.



# Bibliografia

- [1] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.
- [2] Herman H. Goldstine. *The Computer: from Pascal to von Neumann*. Princeton University Press, Princeton (NJ), 1972.
- [3] Jeffrey Ullman. *Principles of database and knowledge base systems*. Computer Science Press, Rockville (MD), 1988.
- [4] E. Codd. A relational model for large shared data banks. *Communications of the Association for Computing Machinery*, 13:377–387, 1970.
- [5] Morton M. Astrahan and Donald D. Chamberlin. Implementation of a Structured English Query Language. *Communications of the Association for Computing Machinery*, 18:580–588, 1975.
- [6] Michael “Monty” Widenius and David Axmark. *MySQL Reference Manual: Documentation from the Source*. O’Reilly Community Press, Cambridge, (MA), 2002.

