Sunto di aritmetica binaria ed architettura del calcolatore

Revisione 1.01

Alessandro Barenghi

September 19, 2019

Questo sunto delle lezioni ha lo scopo di fornire un supporto allo studio *aggiuntivo* rispetto alle lezioni frontali, senza avere la pretesa di sostituirle integralmente. Il testo verrà aggiornato aggiungendo via via i contenuti trattati a lezione. Per quanto sia rivisto con cura, la natura umana del docente fa sì che possano essere presenti errori: non esitate a segnalarli nel caso ne troviate, verranno corretti quanto più in fretta possibile[1]. Per i curiosi, il presente documento è realizzato con LaTeX, ed è rilasciato sotto licenza Creative-Commons Attribution-NonCommercial-ShareAlike 4.0.
`http://creativecommons.org/licenses/by-nc-sa/4.0/`

[1] ovviamente le segnalazioni di falsi positivi non saranno punite in alcun modo, non preoccupatevene

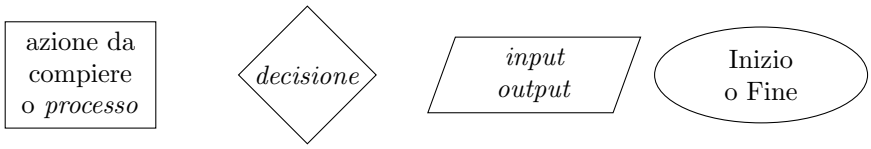

Figure 1: Alcuni dei blocchi usati in un diagramma di flusso

1 Diagrammi di flusso

Tra i formalismi disponibili per la rappresentazione di un algoritmo, uno dei più semplici è quello del diagramma di flusso (flowchart). In un diagramma di flusso viene l' esecuzione di un algoritmo viene suddivisa in una serie di fasi, ognuna delle quali è rappresentata graficamente da un blocco. I blocchi sono collegati da frecce che indicano l' ordine di esecuzione degli stessi.

Sebbene un significativo numero di tipi di blocchi siano stati standardizzati nel documento ISO 5807:1985, per i nostri scopi saranno sufficienti quelli rappresentati in Figura

2 Rappresentazione dei dati in un calcolatore

Un calcolatore moderno elabora i dati utilizzandone una rappresentazione digitale, ovvero rappresentandoli come sequenze *finite* di simboli di un alfabeto anch' esso *finito*. É quindi necessario rappresentare qualunque dato che debba essere elaborato sotto forma di sequenze finite di simboli presi da un alfabeto finito.

A seconda di cosa si desidera rappresentare, è necessario utilizzare una codifica (=convenzione di rappresentazione) opportuna per poterlo fare elaborare dal calcolatore.

L' alfabeto utilizzato per la quasi totalità delle applicazioni è quello binario, ovvero quello costituito da due simboli, 0 e 1.

2.1 Numeri naturali ($\mathbb{N}$)

Il caso dei numeri naturali è il più semplice, essi vengono rappresentati direttamente convertendoli in base 2, ovvero in codifica binaria. Un numero codificato in binario è rappresentato da una sequenza di n cifre $(a_{n-1}a_{n-2}\dots a_1a_0)_2$ che possono essere solamente 0 o 1.

Il significato della codifica è:

$$(a_{n-1}a_{n-2}\dots a_1a_0)_2 = 2^{n-1}\cdot a_{n-1} + 2^{n-2}\cdot a_{n-2} + \dots 2^1\cdot + a_1 + 2^0\cdot a_0$$

Una cifra in codifica binaria è detta *bit*, contrazione di *binary digit*, cifra binaria, appunto. Per quanto possa sembrare poco familiare di primo acchito, non è altro che una variazione sulla comune codifica dei numeri a cui siamo abituati, quella decimale (base 10), dove un numero $(a_{n-1}a_{n-2}\dots a_1a_0)_{10}$ indica:

$$(a_{n-1}a_{n-2}\dots a_1a_0)_{10} = 10^{n-1}\cdot a_{n-1} + 10^{n-2}\cdot a_{n-2} + \dots 10^1\cdot + a_1 + 10^0\cdot a_0$$

+	0	1
0	0	1
1	1	10

(a) Addizione

×	0	1
0	0	0
1	0	1

(b) Moltiplicazione

Table 1: Tabella additiva e moltiplicativa in base 2

Abbiamo familiarità con il fatto che il massimo numero rappresentabile con n cifre decimali sia $10^n - 1$: ad esempio, il massimo numero rappresentabile con 4 cifre decimali è $10^4 - 1 = 10000 - 1 = 9999$ Similmente, osservando il significato della codifica binaria nell'equazione 2.1, si vede come il massimo numero naturale rappresentabile con n cifre binarie sia $2^n - 1$. É possibile utilizzare tutti gli algoritmi per il calcolo a precisione multipla che conoscete anche in codifica binaria, avendo l'accortezza di memorizzare le "tabelline" per la codifica binaria riportate in tabella 1.

Il passaggio da decimale a binario può essere fatto con il consueto algoritmo delle divisioni successive: è sufficiente continuare a dividere per 2 il numero da trasformare annotando il valore dei resti delle successive divisioni in ordine inverso. La sequenza di resti è la codifica binaria del numero.

2.2 Numeri relativi ($\mathbb{Z}$)

Ci sono più strategie possibili per rappresentare gli interi con segno in un calcolatore:

1. Modulo e segno: in questo caso il numero è rappresentato su n bit, usandone $n-1$ per il valore assoluto e uno per il segno (convenzionalmente il bit di segno a 1 rappresenta il $-$, a 0 il $+$). I numeri rappresentabili vanno da $-(2^{n-1} - 1)$ a $(2^{n-1} - 1)$.

    - Vantaggi:
        - La codifica e decodifica del numero convertendo dal decimale è molto semplice
    - Svantaggi:
        - La gestione delle operazioni aritmetiche è va fatta gestendo esplicitamente tutti i possibili casi di segno degli operandi
        - La rappresentazione dello zero è ridondante : viene rappresentato sia come $+0$ che come -0

2. Complemento a 2: il numero a di n bit viene rappresentato codificandolo, se positivo, in maniera naturale, se negativo codificando $2^n - a$. I numeri rappresentabili vanno da -2^{n-1} a $(2^{n-1} - 1)$.

    - Vantaggi:
        - La codifica fa sì che applicando il comune algoritmo di somma a due numeri in complemento a 2, il risultato *se rappresentabile*, venga calcolato correttamente indipententemente dal segno di entrambi

Codifica binaria	Valore interpretato come ...	
	Modulo e Segno	Complemento a 2
011	3	3
010	2	2
001	1	1
000	0	0
111	-3	-1
110	-2	-2
101	-1	-3
100	-0	-4

Table 2: Tabella esemplificativa della codifica su $n = 3$ bit di numeri relativi.

  - É possibile cambiare di segno un numero in modo molto efficiente con le seguenti osservazioni: i) notare che $2^n - a = 2^n - a - 1 + 1 = ((2^n - 1) - a) + 1$ ii) notare che sottrarre qualcosa a $2^n - 1$, che è fatto di soli 1 significa semplicemente prendere un numero e scambiare gli uni con gli zeri, di conseguenza, per fare $(2^n - 1) - a$ è sufficiente scambiare gli uni con gli zeri e viceversa nella codifica binaria di a. Di conseguenza, il cambio di segno viene effettuato cambiando tutti i bit di a e sommando 1 al risultato.
- Svantaggi:
  - La codifica può sembrare meno "naturale" se osservata direttamente

A conseguenza della maggiore efficienza e semplicità nel meccanizzare il calcolo, tutti i calcolatori moderni codificano i numeri relativi in complemento a 2. In Tabella 2 trovate una comparazione del significato della codifica di interi relativi su 3 bit in modulo e segno/complemento a 2

2.3 Numeri razionali ($\mathbb{Q}$)

L' unico modo di codificare esattamente un numero razionale su un calcolatore è di codificare il suo numeratore e denominatore come due interi relativi. É possibile riusare le codifiche viste in precedenza.

2.4 Numeri reali ($\mathbb{R}$) e complessi ($\mathbb{C}$)

Data la densità[2] dei numeri reali, non è possibile codificare neppure un intervallo finito di essi, e.g. $[0, 1]$ con un numero finito di bit. La soluzione a cui si ricorre è una codifica che introduce necessariamente un' approssimazione nel numero rappresentato. Le strategie per gestire i segni usate in precedenza, possono essere utilizzate con profitto anche con i numeri reali e complessi Due strategie possono essere usate:

1. Codifica a virgola fissa: il numero reale viene codificato su n cifre binarie, di cui i codificano la parte intera e f quella frazionaria. Il metodo

[2]da intendersi come la proprietà per cui tra due numeri reali è sempre presente un terzo numero reale, strettamente compreso tra essi

per trasformare un numero decimale in codifica a virgola fissa è identico a quello usato per i numeri naturali, con la differenza che si arresta il procedimento quando si è ottenuto il numero di cifre desiderato e il resto della divisione è da considerarsi senza la parte frazionaria a ogni passo.

- Vantaggi:
  - L' operazione di codifica è semplice ed è effettuabile in modo efficiente
  - Le operazioni aritmetiche possono essere effettuate dalle stesse unità che fanno quelle per i numeri interi
  - L' errore di codifica è fissato per tutti i numeri ed è minore di 2^{-f}
- Svantaggi:
  - L' intervallo di numeri rappresentabili è piuttosto ridotto: va da 0 a $2^i - 1$, nel caso di numeri senza segno e da $-(2^{i-1} - 1)$ a $2^{i-1} - 1$ per quelli in complemento a 2.

2. Codifica a virgola mobile: il numero viene rappresentato codificando separatamente mantissa su m bit ed esponente su e bit, sostanzialmente in maniera simile alla notazione scientifica. Lo standard internazionale per la codifica dei numeri a virgola mobile prevede che il segno viene codificato su un bit separato.

   - Vantaggi:
     - L' intervallo di numeri codificabile è molto più ampio: va da $-(2^m - 1)^{2^e-1}$ a $(2^m - 1)^{2^e-1}$
   - Svantaggi:
     - Le operazioni aritmetiche devono gestire mantissa ed esponente in modo opportuno, sono necessari algoritmi diversi da quelli per la virgola fissa
     - L' errore di codifica è diverso a seconda della grandezza del numero: intuitivamente, errori di codifica sulla mantissa con esponenti più grandi danno origine a errori assoluti più grandi
     - Problema dell' *underflow*: attorno allo zero è presente una zona non rappresentabile, dovuta alla dimensione finita dell' esponente

A causa dell' intervallo di rappresentazione molto ampio, è comune utilizzare la codifica a virgola mobile, per quanto sia necessaria cura nel controllo degli errori di approssimazione. La codifica a virgola fissa offre un' alternativa molto efficiente nel caso non si abbia particolari necessità di intervallo di rappresentazione alto, o si necessiti assolutamente di un errore di approssimazione indipendente dalla dimensione del numero.

2.5 Testo

Nel caso del testo, il problema è quello di codificare una serie di simboli arbitrari, l' alfabeto, in binario. Una volta codificata ogni simbolo, la codifica delle parole si può ottenere giustapponendo quella dei simboli, esattamente come viene fatto nel testo comune.

Considerato che non c'è nessun vincolo semantico a priori (ovvero, non c'è una buona ragione per rappresentare "A" con $(101010)_2$ per quel che vale), in linea di principio ogni tabella di corrispondenze tra i simboli del testo e dei numeri in codifica binaria è buona. L' uso razionale delle risorse del calcolatore[3] ha imposto due criteri nello scegliere questa codifica: i) minimizzare il numero di bit usati per ogni simbolo, ii) avere uno standard comune.

Il punto di convergenza è lo standard ASCII, che rappresenta ogni carattere stampabile come un intero senza segno in codifica binaria su $n = 7$ bit. Potete reperire una tabella con le corrispondenze a questo link: `https://en.wikipedia.org/wiki/ASCII#ASCII_printable_code_chart`, oppure digitando `man ascii` dal terminale della macchina virtuale che usate per gli esercizi.

Un punto degno di nota dello standard ASCII è che la codifica è stata scelta in maniera tale per cui i numeri che codificano le lettere maiuscole sono tra loro contigui e ordinati come le lettere. Ad esempio, il carattere `A` è codificato da $(65)_{10} = (1000001)_2$, `B` da $(66)_{10} = (1000010)_2$ e così via. Lo stesso vale anche per le lettere minuscole, avendo cura di ricordare che la codifica della `a` è $(97)_{10} = (1100001)_2$.

Lo standard ASCII consente anche di codificare i caratteri che rappresentano le cifre di un numero, quando questo è scritto in un testo. Dal punto di vista della codifica ASCII, il carattere `0` viene trattato esattamente come ogni altro carattere, e.g. `a`, e dunque assegnato a un numero intero su 7 bit ($(48)_{10}$, per la precisione). Abbiate cura di non confondere la codifica di una serie di caratteri rappresentanti le cifre di un numero in un testo, con la codifica di quel numero naturale.

Lo standard ASCII ha incontrato alcune limitazioni al momento di rappresentare alfabeti molto ampi, e diversi da quello latino: diverse proposte sono state adottate, tra cui la più comune è UTF-8 che mantiene la stessa codifica per i caratteri stampabili di ASCII, e utilizza più bit per rappresentare i restanti caratteri (fino a 16).

3 Codifica di segnali

La necessità pratica di rappresentare fenomeni naturali in un calcolatore (ad esempio con lo scopo di effettuare esperimenti simulati), richiede di trovare una codifica a valori che sono di per loro natura espressi con numeri reali o complessi. Considerato che è necessario rappresentarli con un numero finito di bit, si avrà una perdita di informazione in questa codifica.

Allo scopo di acquisire misure di una grandezza naturale si utilizzano dispositivi che convertono questa grandezza in una differenza di potenziale, detti sensori. Un esempio molto comune è un microfono piezoelettrico (come quello contenuto in un cellulare), che converte la pressione istantanea esercitata su di esso dall' onda sonora, in una differenza di potenziale. La conversione in segnale elettrico rende possibile la successiva acquisizione del segnale. Restano due problemi per poter acquisire il segnale: esso è continuo nel tempo e (sostanzialmente) continuo nella differenza di potenziale. Dovendo rappresentare l' entità del segnale con un numero finito di bit, così come l' istante temporale in cui esso è stato acquisito, il segnale viene *campionato* e *quantizzato*. Il processo di

[3] in fondo, ogni bit diventerà un pezzo di metallo percorso da corrente, non vale la pena di sprecarli

campionamento prevede il fatto che l' entità del segnale sia memorizzata con una cadenza regolare, detta periodo di campionamento, come un numero, detto appunto campione. L' inverso del periodo di campionamento è detto frequenza di campionamento. La quantizzazione prevede che l' entità del segnale sia rappresentata con un numero finito di bit (tipicamente come un intero con segno). Esempio pratico: quasi tutti i telefoni cellulare campionano il segnale del microfono a 44100 Hz, salvando l' entità del segnale come un intero con segno in complemento a 2 di 16 bit. Quantificare l' effettiva perdita di informazione dovuta a campionamento e quantizzazione è l' oggetto di studio della teoria dei segnali. Il componente elettronico che si occupa di quantizzazione e campionamento (tipicamente è un unico componente integrato) è detto convertitore analogico-digitale (o Analog-to-Digital Converter, ADC).

Una volta campionato e quantizzato, il segnale può essere trattato come una sequenza numerica, e su di esso possono essere effettuate le trasformazioni desiderate. Dopo la sua elaborazione, il segnale elaborato può essere riprodotto, per mezzo di un convertitore digitale-analogico (digital-to-analog converter, o DAC) e un opportuno apparato. Ad esempio, il segnale campionato e quantizzato da un cellulare può essere riprodotto dall' altoparlante del cellulare stesso, che è collegato a un DAC.

3.1 Rappresentazione delle immagini

Sono possibili due approcci, completamente diversi per la loro natura, per rappresentare immagini in un calcolatore.

Formato vettoriale Una immagine in formato vettoriale viene rappresentata in un calcolatore dandone una descrizione sintetica delle forme. Ad esempio, è possibile rappresentare una circonferenza memorizzandone le coordinate dell' origine e la lunghezza del raggio, in un opportuno sistema di riferimento, come 3 numeri, codificati con la precisione desiderata. Il vantaggio della memorizzazione in formato vettoriale è che si ha una rappresentazione esatta della forma desiderata, a meno della precisione dei parametri della stessa. Considerando l' esempio, la circonferenza rappresentata in formato vettoriale ci consente di calcolare la posizione di quanti punti vogliamo di essa, e con la precisione che preferiamo (compatibilmente con le risorse di calcolo). Lo svantaggio principale di un formato vettoriale è che, con rare eccezioni, mal si applica alla rappresentazione di immagini provenienti da fenomeni naturali (e.g., fotografie). In generale, infatti, è difficile andare a derivare una descrizione sintetica (e.g. un'equazione) che descriva la forma esatta di ogni elemento di una fotografia.

Formato raster La rappresentazione raster di un'immagine utilizza la stessa strategia usata per segnali quali il suono, tenendo però conto del fatto che un'immagine è bidimensionale. Dovendola rappresentare come una sequenza di valori è necessario dapprima partizionare l' immagine in una serie di elementi, detti comunemente *pixels* (contrazione di *picture elements*), di prassi quadrati o rettangolari, all' interno dei quali si assume il colore sia costante. Le dimensioni di un immagine sono tipicamente espresse in termini della sua lunghezza e larghezza in numero di pixels. Nel caso si voglia andare a riprodurre un'immagine digitalizzata a grandezza naturale, la sola informazione legata alle

sue dimensioni non è sufficiente: il numero di pixel per riga o colonna infatti non indica a quale dimensione reale corrisponde l' elemento dell' immagine. A questo scopo, si indica con *risoluzione* il numero di pixel per unità di spazio: se i pixel sono quadrati, un' unica misura della risoluzione è sufficiente. Nella pratica, la risoluzione è comunemente espressa in termini di punti per pollice (dots per inch), o punti per centimetro. La suddivisione di un immagine in pixels e il considerare il colore uniforme all' interno del pixel sono a tutti gli effetti la generalizzazione a due dimensioni dei processi di campionamento e quantizzazione.

Allo scopo di rappresentare il contenuto di un pixel, si utilizza una opportuna convenzione per rappresentare i colori (tipicamente salvando 3 valori interi corrispondenti a quanto è intensa la componente rossa, verde e blu) oppure l'intensità luminosa del singolo pixel (tipicamente nel caso di immagini in scala di grigi, con un solo valore).

Scelto il modo di rappresentare il contenuto del pixel, si sceglie una strategia di scansione dell' immagine, ovverosia un ordine in cui le rappresentazioni dei pixel verranno salvate. Ad esempio, un ordine possibile è quello dato dal partire dall' angolo in alto a sinistra, salvando i pixel sulla stessa riga in modo consecutivo. Terminata la prima riga, si riprende salvando da sinistra a destra i pixel della seconda, e così via fino al termine dell' immagine.

4 Cenni di architettura del calcolatore e sistemi operativi

Questa sezione fornisce un sunto della struttura dell' architettura di un calcolatore e del sistema operativo. Nella descrizione sono state operate semplificazioni rispetto alle architetture moderne, senza alterare i principi di funzionamento generali.

4.1 Unità di misura

Considerato che l' informazione nei calcolatori è manipolata e stoccata in codifica binaria, l' unità fondamentale di misura della quantità di informazione è la singola cifra binaria, il **bi**nary digi**t** [b]. Per ragioni di praticità (= un solo bit è poco per codificare un dato significativo) è comune in informatica utilizzare un pacchetto di 8 bit alla volta: con un gioco di parole, 8 bit sono chiamati *byte*[4] [B]. Meno comune, ma a volte usato, è il `nibble`, un insieme di 4 bit contigui. I prefissi e suffissi standard del sistema internazionale (k,M,G,T...) si applicano sia a bit che a byte, ottenendo i multipli opportuni, e.g. un megabyte [MB] è una quantità di informazione pari a 10^6B.

Una nota di carattere pratico è l' osservazione che $2^{10} = 1024 \approx 10^3$, che ci consente di dire che 2^{10}B$\approx$ 1 kB. L' errore causato da questa approssimazione diventa via via più significativo al crescere delle dimensioni dei numeri, fino a toccare il 10% approssimando 2^{40} con 10^{12}.

Per facilitare dunque i calcoli sono stati standardizzati i prefissi per indicare esplicitamente che si stanno indicando potenze di 2 e non di potenze di 10 con un prefisso. I prefissi nuovi aggiungono una "i" minuscola a quelli classici, pertanto

[4]bit significa piccolo morso in inglese, bite morso vero e proprio

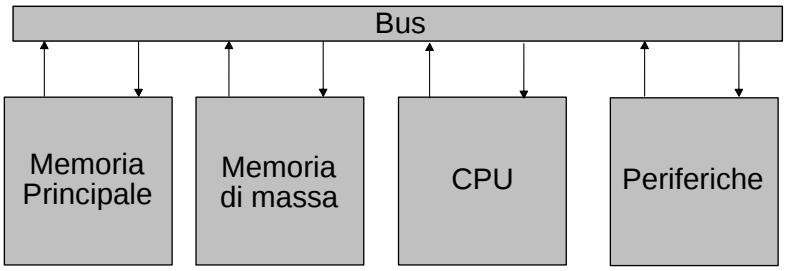

Figure 2: Un semplice schema di una macchina calcolatrice universale con architettura di Von Neumann

un kiB corrisponde a 1024 B, un MiB a 2^{20} B e così via. É utile ricordare che essere in grado di scrivere la prima potenza di 2 che maggiora la quantità che si vuole rappresentare in binario consente di calcolare facilmente il numero di bit necessari per farlo.

4.2 Macchina di Von Neumann

I moderni calcolatori hanno come modello fondamentale quello proposto da Von Neumann (1945): nonostante la loro significativa evoluzione, resta ancora valido per comprenderne il funzionamento. Il modello in questione, schematizzato in Figura 2 è formato da quattro componenti fondamentali comunicanti tramite un un canale comune, il bus (praticamente implementato con degli opportuni cablaggi). A causa delle limitazioni dovute alla presenza di un solo bus, i calcolatori moderni sono tipicamente caratterizzati da più bus, uno per ogni scopo (e.g. trasportare dati, inviare comandi) in modo da evitare l' eccessivo traffico su uno solo di essi. Allo stesso modo, le periferiche sono collegate tramite bus appositi alla macchina, ad esempio l' Universal Serial Bus, o USB.

I componenti del modello di von Neumann sono:

- Central Processing Unit (CPU) o processore: Dispositivo che effettua operazioni aritmetiche, logiche e controlla il comportamento della macchina
- Memoria principale: memoria contenente il codice (= rappresentazione binaria delle istruzioni date alla macchina) del programma da eseguire e i dati ad esso relativi
- Memoria di massa: memoria per lo stoccaggio a lungo termine di dati, non è possibile eseguire direttamente programmi da qui
- Periferiche: dispositivi di supporto per inserire/emettere dati da/verso l' utilizzatore (e.g. tastiera, schermo, microfono)

Analizziamo nel dettaglio i componenti.

Memoria Centrale E' il componente che stocca i dati e i programmi correntemente in esecuzione da parte del calcolatore: la tecnologia con cui è realizzata consente di leggerla e scriverla accedendoci in un qualunque ordine (= anche non in sequenza): viene spesso detta memoria ad accesso casuale o memoria RAM (= Random Access Memory). Fa eccezione una piccola parte di essa, che è solo

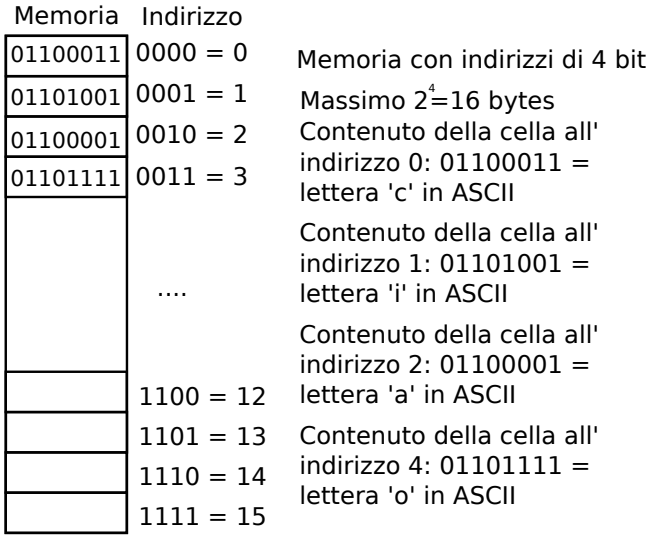

Figure 3: Schema esemplificativo di indirizzamento della memoria principale, con indirizzi da 4 bit

leggibile, e contiene il programma di bootstrap: il primo che il calcolatore esegue per portarsi in uno stato funzionante (= inizializzare le varie componenti). Non venendo scritta di consueto questa piccola porzione è nota come memoria ROM (= Read Only Memory), mentre il programma di inizializzazione è noto come Basic Input/Output System (BIOS).

Per accede alla memoria principale, la si considera logicamente come una sequenza di bytes consecutivi, caratterizzati da un' etichetta numerica, l'*indirizzo*. Tutti i bytes contenuti in memoria hanno quindi una posizione indicata dal loro indirizzo, rappresentato come un numero naturale[5] e, chiaramente, codificato in binario all' interno della macchina.

Essendo necessario avere un indirizzo per ogni byte di memoria principale, dobbiamo scegliere su quanti bit codificare gli indirizzi. Questa scelta limita la quantità massima di memoria a cui possiamo accedere: un indirizzo da k bit $\rightarrow$ ci consente di indirizzare al massimo 2^k bytes di memoria (p.e., indirizzi a 32 bit $\rightarrow$ massimo $2^{32} = 4$ GiB di memoria). La Figura 3 riporta un esempio di memoria da 16 bytes, indirizzata con indirizzi a 4 bit.

Da un punto di vista tecnologico, le attuali memorie principali sono realizzate in modo tale da consentire un tempo d' accesso piuttosto basso, $\sim 60 - 100$ns. Da un punto di vista della capienza, è comune avere da 4 a 32 GiB in un calcolatore nuovo per uso personale e da 512 MiB a 2 GiB a bordo di un cellulare. Un punto chiave della attuale memoria principale è che essa è *volatile*: nel momento in cui si spegne il calcolatore, il suo contenuto viene perso velocemente (in meno di un secondo a temperatura ambiente, il freddo può allungare i tempi a qualche decina di secondi). Per i lettori più interessati: la tipologia più comune è realizzata tramite un circuito integrato, contenente elementi in grado di immagazzinare carica elettrica i.e. dei condensatori di dimensioni molto ridotte. A causa della costante tendenza a scaricarsi di questi, la RAM viene "rinfrescata

[5] in gergo, si dice che la memoria è indirizzata al byte

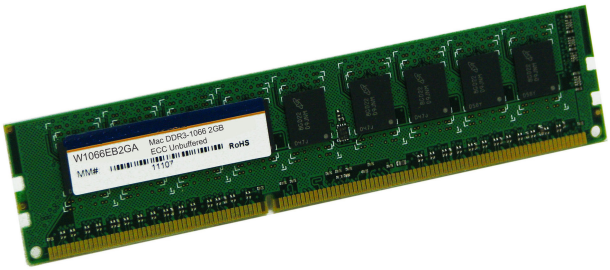

Figure 4: Esempio di memoria RAM moderna: le parti nere sono le capsule plastiche contenenti i circuiti integrati, la parte verde è un supporto meccanico che integra i contatti per poterli collegare al bus

periodicamente" da un circuito apposito, è pertanto detta Dynamic RAM, o DRAM. In Figura 4 è riportata un banco di DRAM di quelli che potete trovare nei vostri calcolatori fissi (è simile a quella di un laptop, semplicemente la parte di adattatore meccanico è più grossa).

Central Processing Unit (CPU). La CPU è il componente effettivamente attivo dell' intero calcolatore, ovvero quello che effettua i calcoli interpretando una sequenza di istruzioni codificate in binario, presenti in memoria principale. In Figura 5 è presente uno schema sommario dell' architettura di una CPU, di cui andiamo ad analizzare i componenti. Il primo componente da esaminare è l' insieme dei *registri* ad uso generale (General Purpose Registers o GPRs): si tratta di un insieme di piccoli componenti di memoria (la loro dimensione tipica è di 4 B o 8 B e il loro numero varia da 4 a 32), che vengono utilizzati dalla CPU per contenere i valori su cui sta effettuando delle elaborazioni (e.g. operazioni aritmetiche). Essi sono tipicamente contraddistinti da un nome simbolico, e.g., `GPR0`, `GPR1`, ... `GPRn`, che viene rappresentato nella macchina con un numero intero. Il secondo componente da prendere in considerazione è l'*Unità Aritmetico-Logica (ALU)*: questa è la porzione del processore dedita effettivamente ai calcoli: può leggere e scrivere il contenuto dei GPR per ottenere i valori su cui effettuare i calcoli e stoccare il risultato. L' attività della CPU è regolata dall' unità di controllo (UC): essa è il componente che pilota le attività della ALU, e richiede caricamenti e salvataggi di dati da memoria, nonchè interpreta l' esecuzione corrente. Per consentire questo, l' UC considera il contenuto di alcuni registri speciali:

- Il registro istruzioni (Instruction Register, IR): esso contiene la rappresentazione binaria dell' istruzione che deve essere eseguita correntemente. Per i lettori interessati: un' istruzione è codificata utilizzando un insieme di bit per descrivere cosa deve fare (e.g. una somma), e i restanti per descrivere su quali registri deve andare ad operare. la porzione di bit dedicata a descrivere la natura dell' istruzione è comunemente detta *codice operativo*, o *opcode*, mentre i registri che usa sono detti operandi dell' istruzione.
- Il contatore di programma (Program Counter, PC): esso contiene l' indirizzo di memoria dove si trova la prossima istruzione da eseguire

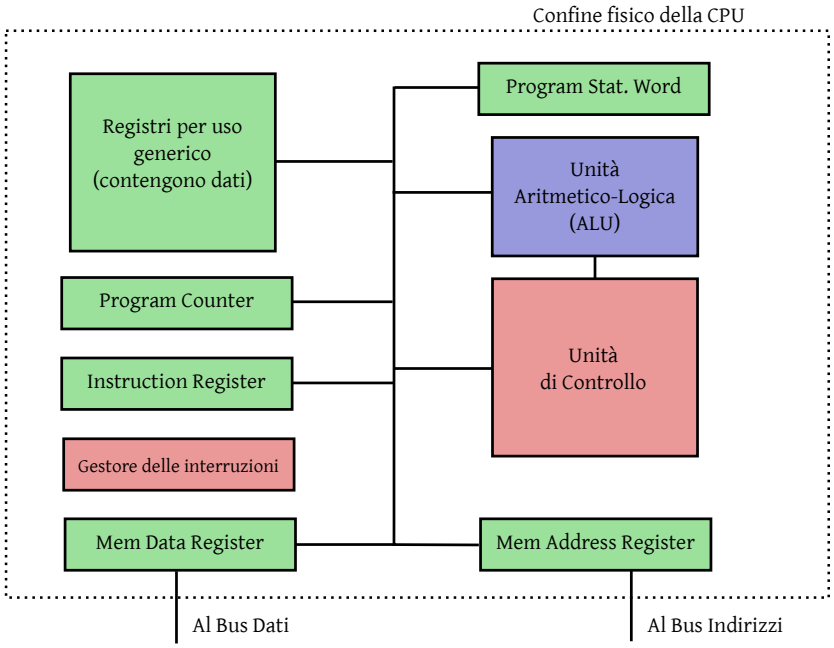

Figure 5: Schema semplificato di una CPU: in verde gli elementi di memoria al suo interno, in rosso quelli che si occupano di regolamentarne il funzionamento, in blu quelli che effettuano la computazione.

- La parola di stato del programma (Program Status Word, PSW): questo registro particolare è da considerarsi come un insieme di registri da 1 bit, che memorizzano le proprietà del risultato dell' ultima istruzione eseguita. Ad esempio, uno dei bit della PSW viene posto a 1 nel caso il risultato precedente sia nullo, ed è detto pertanto *zero-bit*. Analogamente, un altro bit segnala se il risultato dell' ultima istruzione eseguita è negativo.

- Il registro indirizzi e il registro dati verso la memoria (Memory Address/-Data Register, MAR/MDR): questi due registri contengono il valore dell' indirizzo in cui un operazione di memoria deve essere effettuata e il dato corrispondente. Nel caso di una lettura (caricamento) da memoria, il MAR contiene l' indirizzo da cui leggere, l' MDR, alla fine dell' operazione di lettura, conterrà il dato letto. Nel caso di una scrittura, il MAR contiene l' indirizzo a cui scrivere (salvare) il dato contenuto nell' MDR. Al termine dell' operazione di scrittura, il dato nell' MDR sarà stato scritto in memoria all' indirizzo richiesto (sovrascrivendo il dato precedente).

L' unità di controllo regola il funzionamento del processore ripetendo le seguenti 3 fasi all' infinito:

1. **Fetch.** Nella fase di fetch (=recupero), l' UC richiede il caricamento dell' istruzione che si trova in memoria all' indirizzo contenuto nel PC, e una volta che esso è caricato (=è materializzato nel MDR), lo copia nell' IR. A seguito di questo, incrementa il valore di PC in modo che contenga l' indirizzo dell' istruzione successiva.

2. **Decode.** L' UC analizza il contenuto dell' IR, decodifica quindi il tipo di istruzione da compiere, e i suoi operandi.

3. **Execute.** L' UC segnala all' ALU di compiere l' operazione decodificata, segnalando ai GPR che devono essere letti/scritti opportunamente

Le istruzioni eseguite possono essere suddivise in 3 macrocategorie:

1. Istruzioni computazionali: questa categoria di istruzioni comprende tutte quelle che effettuano un calcolo aritmetico o operazioni logiche su operandi e ne salvano il risultato: e.g. moltiplicare il contenuto di un GPR, per quello di un altro e salvare il risultato in un terzo.

2. Istruzioni di accesso alla memoria: queste istruzioni hanno come scopo il caricare un dato da elaborare dalla sua locazione in memoria a un registro ad uso generale, oppure salvare il contenuto di un registro in memoria.

3. Istruzioni di salto: queste istruzioni hanno come unico effetto il modificare il valore del PC. L' effetto ai morsetti sull' esecuzione è quello di cambiare quindi quale sarà la prossima istruzione ad essere eseguita, causando un "salto" in quella che è la naturale sequenza di esecuzione. Un salto in avanti (= a valori maggiori) rispetto al valore corrente del PC causa evita di eseguire un blocco di istruzioni, mentre un salto all' indietro causa la ripetizione delle istruzioni tra il nuovo valore del PC e il vecchio. Il cambio del PC può essere effettuato sia incondizionatamente (l' operazione è detta di salto incondizionato), oppure basandosi su uno o più valori dei bit della PSW. In questo secondo caso si parla di salto condizionato (*branch*), e l' istruzione viene utilizzata per far prendere alla computazione un percorso diverso a seconda del valore del risultato dell' ultima computazione. É possibile, usando opportunamente un salto condizionato, saltare a una certa porzione dell' esecuzione solo se due valori sono uguali, semplicemente effettuando una sottrazione tra loro due e saltando solo se il bit zero della PSW è acceso.

L' ultima porzione della CPU che osserviamo è il sistema di gestione delle interruzioni hardware: esso consente al processore di interrompere l' esecuzione corrente nel momento in cui è segnalata la necessità di attenzione da parte di un evento esterno (e.g. la pressione di un tasto sulla tastiera). Per fare questo, è presente all' interno del processore una porzione di hardware che contiene le posizioni in memoria del codice da eseguire a seconda del tipo di interruzione avvenuta. Nel momento in cui avviene l' interruzione, il processore salta ad eseguire questo codice, detto *gestore dell' interruzione* o *interrupt handler*. Tipicamente questo codice salva lo stato corrente della computazione (valori nei registri, indirizzo del codice dove è arrivata la computazione interrotta), gestisce l' interruzione (e.g., legge il carattere dalla tastiera e lo salva in memoria principale) e ripristina l' esecuzione della computazione precedente. Per mezzo del meccanismo delle interruzioni hardware è quindi possibile per la CPU rispondere alle esigenze delle periferiche in modo tempestivo e coordinare la comunicazione con esse. Un ulteriore sorgente di interruzioni hardware è data dall' orologio di sistema: questo fornisce interruzioni periodiche per consentire alla CPU di avere una cognizione dello scorrere del tempo.

Figure 6: Esempio di disco elettromeccanico:in evidenza i piatti nichelati (3 in foto) e le testine sulla cui estremità è montato il solenoide di lettura/scrittura (non visibile per ragioni di dimensioni)

Periferiche Le periferiche sono componenti del calcolatore che sono, in linea di principio, non indispensabili al procedere della computazione, ma offrono la possibilità di immettere dati all' utente e di visualizzare dati contenuti nel calcolatore. Esempi classici di periferiche sono: mouse, tastiere, monitor, stampanti, attuatori per la domotica. Esse sono collegate a uno o più bus dedicati verso il processore, e sono in grado di segnalare la loro attività tramite cavi dedicati al sistema di gestione delle interruzioni hardware. Tipicamente contengono una piccola quantità di memoria da cui il processore può leggere o in cui può scrivere. Un esempio tipico è la tastiera, che contiene una piccola memoria da qualche byte in cui vengono stoccati i caratteri, codificati in ASCII, corrispondenti ai tasti premuti, fino a quando il processore non ne recupera il contenuto. Un altro esempio è la stampante, che contiene una memoria dove vengono inviati i valori dei dati che devono poi essere stampati su carta. In generale, l' interazione con le periferiche è sempre mediata dal processore, il quale si fa carico di caricare i dati da loro prodotti (gli input dati alla macchina) in memoria principale e copiare dalla memoria principale nelle loro memorie gli output. Per i lettori più curiosi: i calcolatori moderni sono in realtà caratterizzati da alcuni processori "aiutanti" che si fanno carico di effettuare queste copie autonomamente, a fronte di un solo comando di inizio della copia da parte della CPU. Questo tipo di sistema viene detto Direct Memory Access, o DMA.

Memoria di Massa La memoria di massa è il componente che stocca i dati a lungo termine, ma non può essere usata direttamente dalla CPU per i programmi e i dati in uso. Essa è permanente, non viene cancellata quando il calcolatore è spento e, tipicamente è circa 1000 volte più grossa della memoria principale. La tecnologia per realizzarla varia a seconda delle dimensioni, e delle tempistiche di accesso richieste ad essa; al momento attuale le tecnologie prevalenti sono 2:

- **Supporti magnetici.** La memoria di massa magnetica si basa sull' idea di magnetizzare una piccola porzione di materiale ferromagnetico, usando la direzione del campo per indicare il valore del bit. Il suo formato più comune è quello dei dischi fissi elettromeccanici: si tratta di dischi come quello riportato in Figura 6 che vengono letti e scritti con un piccolo elettromagnete posto in cima a una testina di lettura/scrittura. La loro capacità varia tra 1 e 6 TiB correntemente, e il tempo per accedere a un dato va da 2 a 10 ms a seconda della posizione della testina. L' alternativa con capienza più alta è quella dei nastri magnetici: si tratta di nastri di plastica simili alle vecchie musicassette, ricoperti di ferrite. La ferrite sul nastro viene magnetizzata ed essi possono essere letti sequenzialmente. La capacità dei nastri magnetici supera gli 8 TiB per cassetta, ma i tempi di accesso possono raggiungere le decine di secondi nel caso il dato si trovi alla fine del nastro.

- **Memorie di massa a stato solido (solid state disks, SSD).** Si tratta di circuiti integrati, in grado di conservare una carica per tempi molto lunghi (anni). Possono essere collegati alla macchina attraverso lo stesso bus dei dischi elettromeccanici, oppure attraverso il bus dati USB (le cosiddette chiavette). La loro capacità è minore dei dischi elettromeccanici, ha di recente raggiunto il TiB, in compenso il tempo di accesso si aggira nell' ordine delle decine/centinaia di μs per accessi casuali ai dati, circa 100 volte più veloce dei supporti elettromeccanici.

L' accesso fisico alla memoria di massa avviene con le stesse modalità dell' accesso alle memorie delle periferiche in quanto, in origine i calcolatori non erano dotati di memoria di massa, e andavano riinizializzati da capo ad ogni accensione.

4.3 Il sistema operativo

Fino ad ora abbiamo considerato il caso in cui la macchina sta eseguendo un solo programma alla volta. Questo scenario è effettivamente quello che avviene in piccoli calcolatori dedicati (e.g. quello a bordo di una macchina fotografica), ma nella maggior parte dei casi desideriamo eseguire più di un programma "contemporaneamente" su un calcolatore. A questo scopo è sono stati creati i *sistemi operativi*: l' idea chiave di un sistema operativo è quella di gestire direttamente l' hardware, offrendo a tutti gli altri programmi, detti *applicazioni*, devono essere eseguiti l' illusione di avere la macchina completamente per sè. Di conseguenza, sulla stragrande maggioranza dei calcolatori, è il sistema operativo ad avere il controllo delle operazioni della macchina e a decidere cosa viene eseguito, e quando. In particolare, il codice che va a gestire le interruzioni è parte del sistema operativo.

In particolare, le attività fondamentali del sistema operativo sono:

- *Gestione e protezione della memoria principale*: Allo scopo di evitare interferenze tra programmi diversi, il sistema operativo gestisce l' accesso alla memoria (direttamente, o per mezzo di un componente hardware di supporto, la Memory Management Unit (MMU)). L' effetto netto è che ogni applicazione vede uno spazio di indirizzamento proprio, contiguo, e indipendente da cosa stiano facendo gli altri, mentre il sistema operativo,

o l' MMU si occupano di tradurre gli indirizzi di memoria del programma, detti *indirizzi virtuali*, negli indirizzi fisici in memoria di massa. L' intero processo è trasparente dal punto di vista dell' applicazione, mentre consente al sistema operativo di isolare le zone di memoria fisica acceduta dai vari programmi, consentendo di applicare opportune prassi di sicurezza. Il sistema operativo è anche incaricato di caricare i programmi dalla memoria di massa in memoria principale, nel momento in cui devono iniziare la loro esecuzione, e di rimuoverli una volta che la loro esecuzione è terminata.

- *Scheduling*: Il sistema operativo simula l' esecuzione contemporanea di più programmi, eseguendoli per una limitata porzione di tempo a testa (meccanismo di *time-sharing*). A questo scopo, il sistema operativo conserva una lista di tutti i programmi che sono correntemente che devono essere eseguiti a turno, del punto dove è giunta la loro computazione (indirizzo in memoria dell' istruzione), e i valori dei registri della CPU in quell' esecuzione. Nel momento in cui viene segnalata un'interruzione dall' orologio di sistema, il gestore, che è parte del sistema operativo, viene eseguito e decide a chi tocca l' esecuzione per il prossimo quanto di tempo, tenendo in conto il fatto che nessun programma deve restare fermo a tempo indeterminato (*fairness* dello scheduling). Una volta scelto quale programma deve essere posto in esecuzione, il sistema operativo carica il suo contesto di esecuzione (il contenuto dei registri della CPU) e salta all' istruzione che deve essere eseguita di quel programma. Nel caso, come nelle moderne piattaforme, sia presente più di una CPU, operante in parallelo, il sistema operativo decide, per ognuna di esse, quale applicazione deve essere eseguita.

- *Gestione del filesystem*: Per offrire un accesso alla memoria di massa più organizzato e meno prono a conflitti tra diverse applicazioni dell' accesso diretto, il sistema operativo offre ad esse l' astrazione di filesystem.

- *Gestione delle periferiche*: L' inizializzazione delle periferiche in uno stato stabile (e.g. accendere il video, lasciandolo completamente nero all' inizio, accendere i led sulla tastiera) e la loro gestione durante il funzionamento (leggere/scrivere dati da/verso le loro memorie interne) è un' attività che ha in carico il sistema operativo. I dati e, se necessario, la possibilità di interagire con le periferiche, vengono poi offerti secondo un' opportuna astrazione (ad esempio dei files) alle applicazioni.

Un applicazione può richiedere servizi al sistema operativo per mezzo dell' esecuzione di una *chiamata di sistema* o *syscall*. Una syscall, concretamente, è una porzione del codice del sistema operativo che può essere invocata da un programma affinchè effettui per esso operazioni non direttamente consentite. L' esempio più tipico di syscall è quella che viene effettuata per leggere i caratteri che un utente sta fornendo tramite una tastiera. Al momento dell' esecuzione della syscall, il sistema operativo svolge la funzione per cui è stato invocato e, se il quanto di tempo non è terminato, rimette in esecuzione l' applicazione.

4.4 Filesystem

L' astrazione più comune con cui viene offerto l' accesso alle memorie di massa è quella di *filesystem*. Un filesystem è un modo per organizzare i dati su una memoria di massa che assegna a gruppi di zone della memoria di massa, detti *files*, un nome simbolico che li identifica univocamente (e.g. mio_documento.txt). In pratica un filesystem consiste di un elenco di nomi di files, e delle posizioni dei blocchi di dati sulla memoria di massa. Il gestore del filesystem, parte del sistema operativo, offre alle applicazioni la possibilità di richiedere i dati presenti in un file, utilizzando il suo nome simbolico. Indipendentemente dalla posizione dei blocchi di dati sul disco, il file è presentato all' applicazione come una sequenza contigua di bytes, all' interno della quale l' applicazione può scrivere nella posizione che preferisce. L' accesso al file è effettuato in maniera simile a un nastro: il file viene letto sequenzialmente per mezzo di un cursore che può essere spostato a seconda delle necessità dell' applicazione.

Con il crescere del numero di files nei moderni filesystem, si è reso necessario offrire una forma di organizzazione migliore del solo nome. A questo scopo, i filesystem moderni offrono un' organizzazione gerarchica dei files in *directory*. Le directory si comportano come contenitori e possono contenere gruppi di files o altre directory a loro volta. Questo fa sì che si possa convenientemente rappresentare un intero filesystem sotto forma di un albero di directory e files, simile ad un albero genealogico, dove la relazione padre-figlio tra due elementi indica che il figlio è contenuto all' interno del padre. Il vantaggio più evidente, oltre alla capacità di suddividere i files per aree semantiche (una directory per i file contenenti musica, una per le immagini, una per i programmi ...), è la possibilità di avere files con lo stesso nome, a patto che risiedano in directory diverse. Pertando l' identificativo unico per accedere a un file non è più il solo nome, ma il suo *percorso* (o *pathname*) ovvero una codifica testuale dell' intero percorso per giungere al file, a partire dalla radice dell' albero genealogico. Per rappresentare testualmente il passaggio padre-figlio si usa un carattere eletto a separatore di percorso (*path separator*). La scelta del path separator dipende dal sistema operativo, le più comuni sono

- / in sistemi UNIX-like (Linux, MacOS, FreeBSD)
- \in tutti i sistemi Windows

In sistemi UNIX like esiste un solo albero, la cui radice è codificata in testo con un singolo /, mentre in tutti i sistemi Windows possono esserci più radici, tipicamente una per ogni periferica di massa, e sono identificate da lettere maiuscole, seguite da due punti e un \. Ad esempio, la codifica della radice a cui è assegnata la lettera `C` è C:\.

5 Introduzione alla programmazione

Un linguaggio di programmazione è un insieme di regole per poter esprimere sotto forma di testo (codificato nella macchina in ASCII nel nostro caso), un algoritmo. In maniera simile a quello che accade per i linguaggi naturali, anche un linguaggio di programmazione è caratterizzato da tre livelli a cui si può interpretarlo:

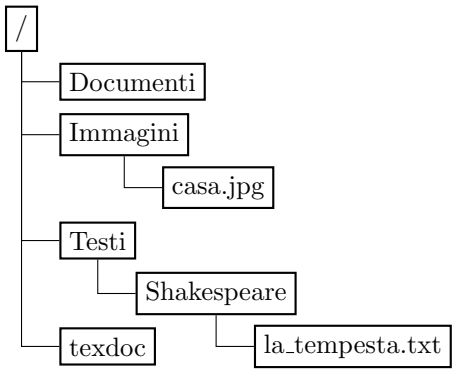

Figure 7: Esempio di filesystem

1. **Lessico.** Il lessico di un linguaggio è l' insieme di regole che caratterizzano la forma delle sue parole, o *lessemi*. Un modo banale di esprimere queste regole è un vocabolario, ovvero una lista di tutte le parole valide del linguaggio. Per contro, è possibile anche esprimere in forma sintetica alcune delle regole che caratterizzano tutti i lessemi. Ad esempio, per l' italiano, tutti i lessemi che finiscono in -gia, con i atona, hanno il plurale che termina in -gie se la g è preceduta da vocale, in -ge se preceduta da consonante. Un esempio di errore lessicale è omettere un accento su una parola che ne necessita. É possibile correggere errori lessicali automaticamente in modo piuttosto efficiente (e.g. il correttore ortografico a corredo di programmi di videoscrittura).

2. **Sintassi.** La sintassi di un linguaggio è l' insieme di regole che stabiliscono come i lessemi vanno combinati per produrre una frase (e.g. quali preposizioni precedano un complemento, l' ordine di soggetto e complemento oggetto in alcune lingue). Il punto fondamentale della sintassi di un linguaggio di programmazione è che essa non deve essere *ambigua*, ovvero, una sequenza di lessemi non deve avere due interpretazioni valide. Per contro, l' ambiguità è un carattere comune dei linguaggi naturali: in italiano la frase "Luigi ha visto un uomo nel parco con il telescopio" presenta un' ambiguità sintattica per cui sono valide due interpretazioni. L' individuazione di errori di sintassi in una frase di un linguaggio, posto che essa non sia ambigua, è fattibile in modo efficiente.

3. **Semantica.** La semantica di un linguaggio è l' insieme di regole che associano a ogni frase un significato, tipicamente come composizione dei significati dei singoli lessemi. L' individuare errori semantici in un testo espresso in un linguaggio di programmazione non è possibile automaticamente nel caso più generale. La conseguenza diretta è che non esiste un modo automatico di dire se un generico programma è corretto o meno (è possibile farlo per alcune classi di programmi).

Una parte di questo corso si occupa di insegnarvi lessico e sintassi dei linguaggi C e Fortran, nonchè la semantica di elementi di base di essi. Comprendere il modo di combinare gli elementi di base in programmi con il valore semantico che volete, ovvero che facciano fare al calcolatore quello che desiderate, è la

seconda parte di questo corso. La combinazione di queste due parti fa sì che siate in grado di programmare, ovverosia di tradurre in un linguaggio di programmazione un algoritmo di vostra concezione.
Per questo scopo, vi serve seguire una serie di passaggi:

0. **Concepire l' algoritmo.** É il passo iniziale in cui inquadrate il problema e lo descrivete a voi stessi, anche informalmente, raffinandone la descrizione fino al punto in cui è suddiviso in operazioni elementari, decisioni, ed eventualmente ripetizioni delle stesse.

1. **Codificare l' algoritmo.** A questo punto potete usare un programma applicativo che vi consenta di scrivere del testo ASCII (e.g. un qualsiasi simil-notepad) per codificare l' algoritmo in un programma nel linguaggio di programmazione che preferite. Il testo ottenuto è comunemente chiamato *codice del programma*.

2. **Compilare il programma.** Il testo da voi prodotto e salvato in un file viene dato in ingresso a un programma, il *compilatore*, il quale ne verifica la correttezza lessicale e sintattica, dopodichè lo traduce in una codifica binaria che è quella che la macchina è in grado di interpretare sia per le istruzioni, che per i dati. Il file risultante da questa traduzione viene detto comunemente *binario eseguibile* o *codice oggetto*.

3. **Avviare il programma.** Una volta ottenuto il binario eseguibile potete dire al vostro calcolatore di iniziare effettivamente la sua esecuzione. Il sistema operativo si farà carico di leggerlo dalla memoria di massa, caricarlo in memoria principale e iniziare ad eseguire i suoi contenuti.

Fatto questo, se la concezione e la codifica del programma sono corrette, avete prodotto il programma che volevate. Purtroppo, non sempre il tutto va come si deve, di conseguenza è necessario avere un modo di ispezionare il funzionamento di un programma mentre sta girando. Per questo sono disponibili dei programmi, detti *debugger*, che prendono in ingresso un *binario eseguibile* e consentono di controllare la sua esecuzione passo passo interattivamente. Ad esempio è possibile interromperla quando esegue un certo passaggio e stampare i valori in memoria. Va notato che è possibile utilizzare un debugger su un qualunque oggetto binario a vostra disposizione: potete quindi esaminare anche i programmi applicativi già presenti sulla vostra memoria di massa, anche se le loro dimensioni non banali potrebbero rendere la comprensione completa un po' faticosa.

Contents