1

PROLOGO

Questo capitolo ha il solo scopo di introdurre in modo semplice alcuni concetti elementari che – a mia esperienza – sono particolarmente utili per chi voglia addentrarsi nel mondo dei calcolatori. Non ho nessuna pretesa di essere esaustivo, né di descrivere accuratamente come funziona esattamente il vecchio computer che un mio attempato zio ha trasformato in una gabbietta per i criceti. Se tu, lettore, vorrai informazioni dettagliate potrai cercarle altrove, ad esempio curiosando via Internet sui nodi dei produttori. Troverai modo di perdere un sacco di tempo. Spero però che queste poche pagine ti possano aiutare ad orientarti. [1]

La distinzione tradizionale – e sostanzialmente sempre valida – è tra *hardware* e *software*.

- *Hardware* (letteralmente: la ferramenta) è, in termini rozzi, la parte concretamente visibile della macchina. Si è partiti con una stanza piena di pannelli su cui erano montati tutti i componenti elettronici necessari e di cavi di collegamento tra i pannelli, passando per una serie di armadi, per arrivare infine alla situazione attuale: praticamente una scatola che contiene delle schede e che all'esterno presenta degli spinotti per il collegamento di oggetti vari, oltre che per la connessione alla presa di

[1] Vorrei anche aggiungere che se sei un lettore esperto ed informato sullo sviluppo dei calcolatori troverai queste note un po' vecchie come contenuto: l'elenco delle caratteristiche dei processori recenti che qui non sono prese in considerazione occuperebbe un numero di pagine certamente superiore a quello di queste note. Il motivo è molto semplice: il destinatario di questi appunti non è un lettore esperto, ma un lettore che abbia la curiosità di guardare per la prima volta oltre la facciata del mouse, dello schermo e dei pacchetti prefabbricati (e spesso costosi). Se appartieni a questa seconda categoria, e il mio lavoro sarà riuscito a solleticare la tua curiosità e ad indurti a cercare altrove informazioni più precise avrò raggiunto la metà del mio scopo. Se poi, col tempo, sarai diventato abbastanza esperto da criticare queste note come troppo semplicistiche, allora potrò ben ritenere di aver pienamente realizzato il mio obiettivo.

alimentazione che fornisce l'energia elettrica. Come dire che la stanza si è ridotta ad una scatola, i pannelli si sono ridotti a poche schede con dei circuiti stampati e i fasci di cavi si sono ridotti a tracce di rame (o comunque materiale conduttore) depositate direttamente sulle schede.

- *Software* (un termine coniato in contrapposizione ad hard–ware) è, in termini altrettanto rozzi, la parte che non è concretamente visibile se non grazie ai suoi effetti: un insieme di programmi che ci permettono di utilizzare il calcolatore.

1.1 Uno sguardo dall'alto sull'hardware

Lo schema tradizionale, e pur sempre valido, è semplice. La macchina contiene tre componenti fondamentali:

1. la *CPU* (Central processing Unit), o processore: il componente che ha la funzione di eseguire le operazioni di elaborazione. In termini molto crudi: quello che svolge effettivamente il lavoro.
2. la *memoria*: il serbatoio di informazioni da cui la CPU attinge istruzioni e dati, e dove vengono memorizzati i dati elaborati. Sempre in termini crudi: il blocco note da cui la CPU legge istruzioni e dati, e su cui scrive i risultati.
3. il *sottosistema di I/O* [2] e le *periferiche*: tutto quanto serve a memorizzare permanentemente programmi e dati, o a comunicare con l'utente finale — un rappresentante dell'umanità.

Lo schema è rappresentato in fig. 1.1. [3] Lo scambio di dati tra CPU, memoria e periferiche avviene tramite un bus, [4] rappresentato in figura dalle frecce bidirezionali. La freccia che collega la memoria con le periferiche è

[2] L'acronimo I/O sta per Input/Output, e fa riferimento a qualunque operazione di trasferimento di dati da o verso un componente del calcolatore.

[3] Una nota da tener ben presente qui e in tutto il seguito. Il calcolatore è una struttura complessa, ma non più di tanto (è pur sempre frutto della nostra intelligenza limitata). Il modo migliore per capirne il funzionamento è considerare dei blocchi elementari che possono a loro volta essere scomposti in altri blocchi, &c. Questo capitolo potrebbe essere un buon esempio. Parti considerando tre blocchi, la CPU, la memoria, le periferiche; poi apri questi blocchi, e scopri che ciascuno di questi è a sua volta un insieme complicato di altri blocchi, che a loro volta... L'importante è non spaventarsi. Il guaio è che le strutture complesse hanno la tendenza a sfuggire di mano a chi le ha costruite: un bel giorno scopri che le regole di funzionamento permettono operazioni – e spesso errori – che il progettista non sognava neppure.

[4] Letteralmente: un autobus, o torpedone — anche se l'immagine dei bit che tutti rigorosamente in fila indiana attendono di salire sull'autobus è un po' pagliaccesca. In pratica un mezzo di trasporto di informazioni costituito fisicamente da un fascio di fili che possono trasferire dei segnali elettrici.

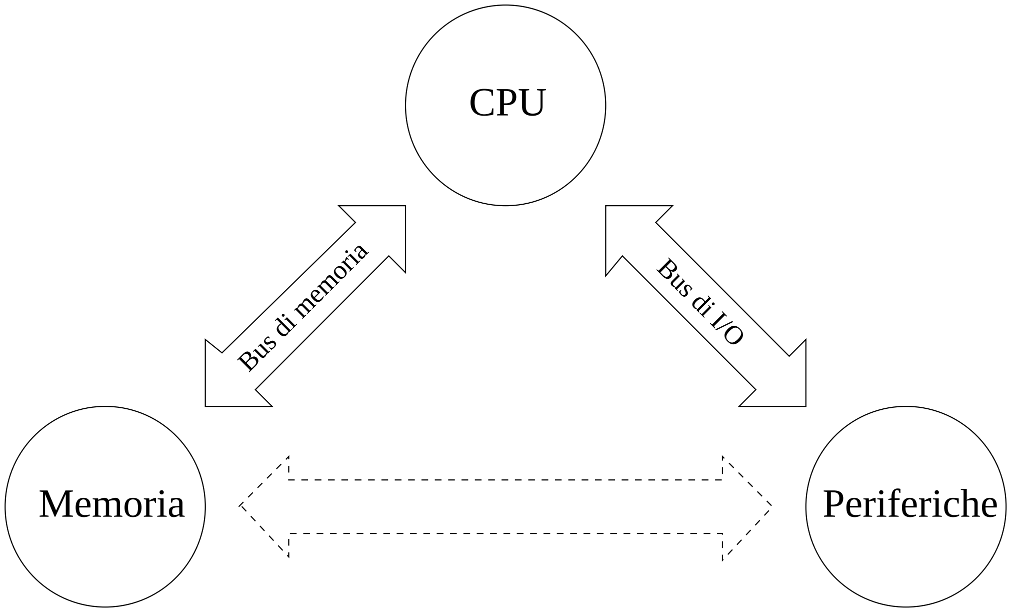

Figura 1.1. Lo schema fondamentale della struttura hardware.

tratteggiata, ad indicare che lo scambio di informazioni è sempre in qualche modo controllato dalla CPU o comunque da qualche componente che si fa carico dell'esecuzione ordinata dei trasferimenti di dati.

1.1.1 La CPU

La CPU, come ho detto, è la parte destinata a svolgere effettivamente le operazioni di elaborazione – potremmo dire: è il cervello del computer. Lo schema fondamentale è rappresentato in figura 1.2. Si possono distinguere in linea di massima cinque componenti.

1. L'unità di controllo: presiede al caricamento ed alla decodifica delle istruzioni, nonché al trasferimento di dati da e verso la memoria.
2. Unità logico–aritmetica, o ALU (*Arithmetic and Logical Unit*): esegue le operazioni logiche ed aritmetiche sui dati. Attualmente è affiancata da una seconda unità detta *Floating Point Processor* o *Coprocessore Matematico* che svolge le operazioni su dati in virgola mobile. [5]

[5] Operazioni logiche sono, ad esempio, il confronto tra due numeri o un test per stabilire il segno di un numero. Operazioni aritmetiche sono, ad esempio, le operazioni elementari di addizione, sottrazione, moltiplicazione e divisione tra numeri interi. Il termine *floating point processor* si traduce letteralmente con *elaboratore di punto fluttuante* – un termine dal significato oscuro e vagamente inquietante al quale preferisco quello di *elaboratore di operazioni in virgola mobile*. Si riferisce alle operazioni aritmetiche su numeri non interi; per intenderci: i numeri con la virgola che tutti abbiamo imparato ad usare alle scuole elementari. Il perché la virgola debba essere mobile lo capirai più avanti, quando ti avrò spiegato come si rappresentano i dati in memoria. Nei

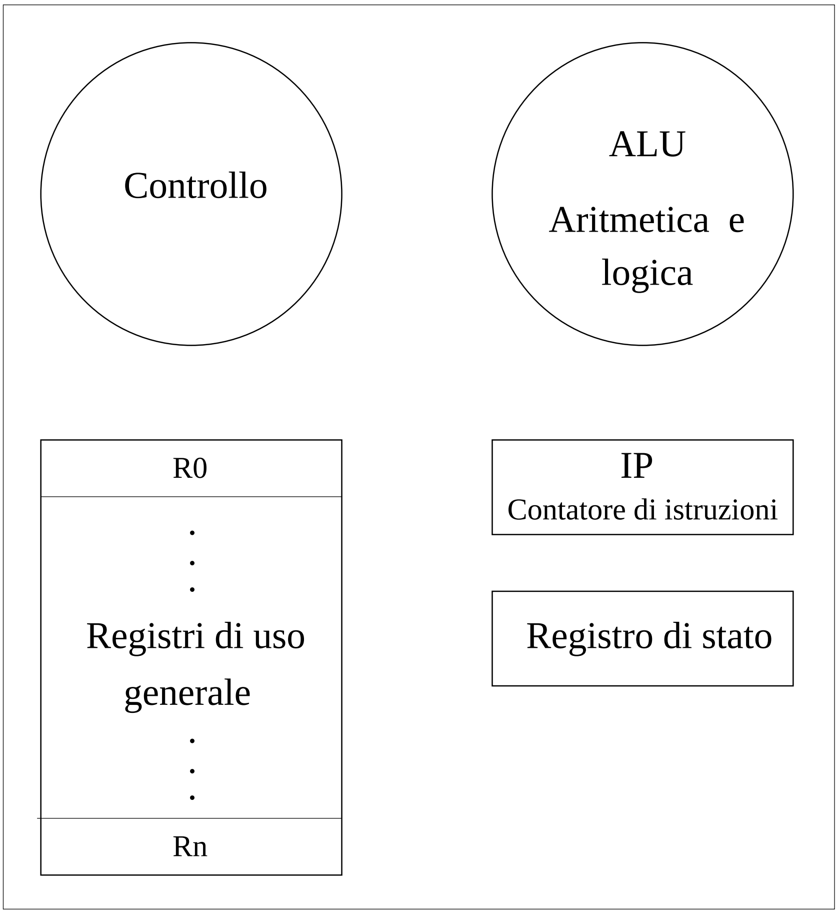

Figura 1.2. Lo schema fondamentale della CPU.

3. I registri generali: possiamo immaginarli come delle celle destinate a contenere le informazioni che devono essere immediatamente disponibili alla CPU per l'elaborazione. L'uso specifico verrà illustrato in maggior dettaglio in seguito.
4. IP, o *Instruction Pointer*: è il registro che controlla l'ordine di esecuzione delle istruzioni di un programma. Anche questo meccanismo verrà discusso in maggior dettaglio più avanti.
5. Il registro di stato descrive, appunto, lo stato corrente della CPU. Una descrizione completa esula dagli scopi di queste note. Un esempio di informazione di stato è il seguente: alla fine di ciascuna operazione aritmetica o logica il registro di stato indica se l'operazione abbia dato risultato positivo, negativo o nullo. Questa informazione risulta preziosa quando

---

primi elaboratori il floating point processor era assente, e tutto il calcolo in virgola mobile veniva svolto con una sequenza più o meno lunga di istruzioni dalla ALU. In seguito è comparso come circuito separato da quello della CPU, che veniva montato solo a richiesta — per motivi di costo. Attualmente è parte integrante della CPU.

si debba decidere in funzione del risultato come proseguire l'elaborazione (se il risultato è negativo allora procedi così, altrimenti...).

1.1.2 La memoria

La memoria, o brevemente RAM, [6] è il deposito di informazioni che la CPU deve elaborare e delle istruzioni che indicano come elaborarle. In modo ingenuo la si può rappresentare come una lunga cassettiera, i cui cassetti sono numerati sequenzialmente; ogni cassetto può contenere un dato. Il numero che contraddistingue il cassetto viene detto *indirizzo*, o *address*; il *dato* contenuto nel cassetto è semplicemente un numero, rappresentato in forma binaria. La figura 1.3 riassume lo schema dell'organizzazione della memoria.

L'unità elementare di informazione viene detta *bit*, risultato della contrazione delle due parole "*bi*nary digi*t*", o cifra binaria. L'aggettivo *binaria* significa che il bit può assumere due stati distinti. [7] Per convenzione si identificano i due stati con le cifre 0 (detto anche stato di bit *clear*) e 1 (detto anche stato di bit *set*). Tra i programmatori italiani si parla talvolta di bit abbassato (0) o alzato (1) — come si trattasse di un bi*t*rillo.

Ai fini dell'indirizzamento i bit vengono raggruppati in sequenze di lunghezza fissa. Sulla lunghezza i costruttori hanno fatto le scelte più svariate, ma attualmente esiste una convenzione adottata pressoché universalmente:

- il campo di lunghezza minima, detto *byte* o talvolta *carattere*, è formato da 8 bit;
- due bytes consecutivi, ovvero 16 bit, formano una *word*, talvolta tradotto letteralmente in italiano come *parola*;

---

[6] L'acronimo RAM significa *Random Access Memory*, ovvero memoria ad accesso casuale, L'aggettivo *casuale* è da intendersi nel senso che in qualunque momento si può accedere a qualunque posizione della memoria, sia in lettura che in scrittura. Esiste anche un tipo di memoria detta ROM, acronimo per *Read Only Memory*, ossia memoria accessibile in sola lettura. Questo tipo di memoria viene utilizzato per mantenere permanentemente le istruzioni che la CPU esegue al momento dell'accensione, e che devono essere protette da modifiche accidentali.

[7] Fisicamente un bit può essere realizzato da un qualunque oggetto che possa presentarsi in due stati distinti: ad esempio, un birillo può essere in posizione verticale o orizzontale, una finestra può essere aperta o chiusa, una lampadina può essere accesa o spenta, un magnetino può essere polarizzato in due direzioni opposte, in un circuito può passare corrente in una direzione o in quella opposta, agli estremi di un circuito può essere applicata una tensione o quella opposta, &c. Fino alla metà degli anni '70 del secolo ormai trascorso le memorie venivano effettivamente realizzate con degli anellini di ferrite che potevano essere magnetizzati in due sensi grazie ad una rete di fili che passavano all'interno. Attualmente si usano circuiti bistabili, nel senso che possono restare stabilmente in due stati distinti.

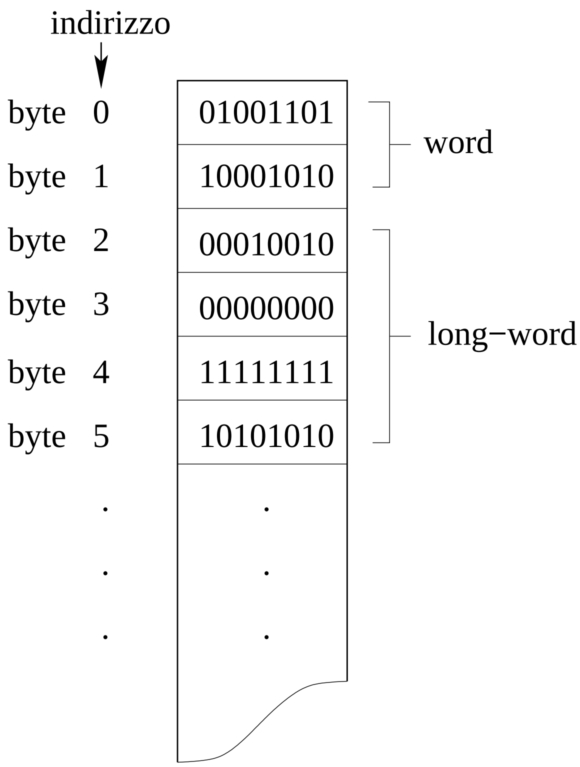

Figura 1.3. Lo schema della memoria: ogni cella (o cassetto, se si vuol usare l'immagine dalla cassettiera) può contenere un'informazione binaria, tipicamente un byte (8 bit). La cella è identificata da un numero: il suo indirizzo. Più celle consecutive possono raggrupparsi in modo da costituire un blocco di informazione che è multiplo di quello della cella fondamentale.

◦ quattro bytes consecutivi, ovvero 32 bit, formano una *long–word*.

Tieni presente però che benché il raggruppamento a 8, 16 e 32 bit sia ormai universalmente accettato non c'è affatto uniformità nell'uso dei termini word e long–word. Ad esempio, è ormai comune in molti manuali trovare il termine *word* per indicare un campo di 32 bit, o 4 bytes, accompagnato dal termine *half–word* per indicare il campo di 2 bytes. Occorre un po' di attenzione prima di avventurarsi nell'interpretazione delle pagine dei manuali.

La quantità di memoria disponibile sull'elaboratore viene indicata in Kilobytes (o Kbytes), Megabytes (o Mbytes) e Gigabytes (o Gbytes). Questi termini ti sembreranno familiari, in quanto ereditati dalla terminologia del sistema metrico decimale. Ma c'è un tranello: un Kilobyte (o Kbyte) non corrisponde a 1000 bytes (come ingenuamente penserebbe chi ha imparato che un kilogrammo corrisponde a 1000 grammi). Il motivo è molto semplice: 1000 è un numero che in base 2 si scrive 1111101000 — molto brutto. Il numero binario 10000000000 (un 1 seguito da 10 zeri, ovvero 2^{10}) è molto più elegante e facile da ricordare, così come nella rappresentazione decimale

1000, ossia 10^3, è più elegante di 1024, ossia 2^{10}. Il trucco perverso sta nel fatto che la differenza tra 10^3 e 2^{10} è decisamente piccola, ... sono quasi uguali. Da qui il termine corrente Kbyte per indicare 2^{10} bytes. Quindi, un Kbyte è formato da 1024 bytes. Analogamente, un Mbyte corrisponde a 2^{20} bytes (= 1 048 576 bytes), che è circa 1 milione di bytes, e un Gbyte corrisponde a 2^{30} bytes (=1 073 741 824 bytes), che non è troppo lontano da 1 miliardo di bytes — anche se la differenza comincia ad avvicinarsi al 10%. Naturalmente, ci si può divertire ad imbrogliare ancor di più le carte parlando, ad esempio, di 1000 Kbytes, che corrispondono a 1 024 000 bytes, e non ad un Mbyte, &c. Sembra uno scherzo, ma il guaio è che qualcuno lo fa davvero. A dimostrazione del fatto che l'informatica rende tutto più semplice!

1.1.3 Il sottosistema di I/O e le periferiche

Qui l'argomento si fa necessariamente molto vasto; per questo cercherò di ridurre le informazioni all'essenziale. Il nome *unità periferica* si riferisce a qualunque apparecchiatura in grado di colloquiare con la CPU, inviandole o ricevendone dati. In modo grossolano possiamo distinguere tre tipi di unità periferiche: [8]

i. quelle destinate a consentire lo scambio di informazioni tra l'uomo e la macchina: la tastiera, il mouse, lo schermo del terminale video, la stampante, le casse acustiche, lo scanner;
ii. quelle destinate allo scambio diretto di informazioni tra elaboratori: le reti di qualunque tipo (cavo seriale, cavo parallelo, cavo coassiale, cavo ottico, radio) ed il modem; [9]
iii. quelle destinate alla memorizzazione di dati che devono essere conservati per un tempo più o meno lungo, tipicamente dischi e nastri, ma ora anche schede di memoria estraibili che vengono trattate a tutti gli effetti come dischi.

---

[8] Che la distinzione sia grossolana è provato, ad esempio, dal fatto che non si saprebbe bene dove collocare oggetti quali le schede e bande perforate. I programmatori con un buon quarto di secolo di attività alle spalle sanno benissimo che per oltre un decennio la banda perforata e le schede sono state di fatto l'unico mezzo per inserire dati e istruzioni nel calcolatore — salvo nei casi in cui si usavano gli interruttori del pannello di controllo per inserire direttamente le istruzioni in forma binaria. Quindi bande e schede perforate si collocherebbero nella prima categoria. D'altro canto gli stessi programmatori sanno altrettanto bene che prima dell'avvento dei dischi le bande e le schede perforate sono state ampiamente usate anche per memorizzare dati destinati ad essere riutilizzati sull'elaboratore — un uso che rientra nella terza categoria. Ma oggetti di questo genere sono comunque obsoleti e dimenticati.

[9] Il nome *modem* è la contrazione di "modulator–demodulator", in quanto ha la duplice funzione di tradurre i dati binari in suoni che vengono inviati su una linea telefonica, e di tradurre i suoni ricevuti in dati binari.

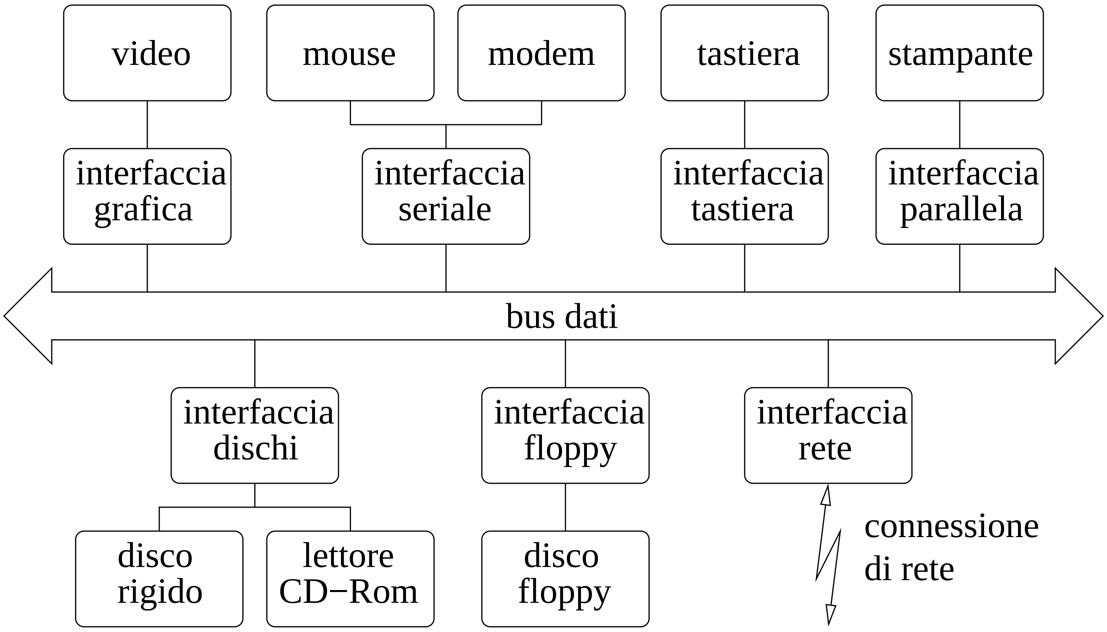

Figura 1.4. Lo schema del sottosistema di I/O.

Di qualunque oggetto si tratti, lo schema fondamentale è unico, ed è illustrato a grandi linee in figura 1.4. Si tratta di una configurazione abbastanza tipica per un computer domestico, fatta eccezione per il collegamento in rete, più frequentemente realizzato in ambienti di lavoro. Le periferiche prese in considerazione sono tastiera, video, mouse, stampante, modem, disco rigido e floppy, lettore di CD-Rom e connessione alla rete.

Il collegamento tra tutte le periferiche ed il resto del sistema avviene attraverso un *bus*. Su questo vengono collegate le interfacce di I/O, che hanno il compito di gestire la trasmissione di informazioni tra bus e unità periferica. Tipicamente occorre un'interfaccia di I/O per ciascun tipo di periferica: gestire la tastiera è faccenda diversa e sensibilmente più semplice che gestire un disco!

Il bus, come ho già detto, è in buona sostanza un fascio di cavetti elettrici (e proprio così si presentava sulle macchine di un paio di decenni fa). Attualmente i cavetti sono sostituiti da linee di materiale conduttore stampate su una scheda, detta *scheda madre*, o *motherboard*. Le interfacce possono essere integrate direttamente sulla scheda madre, oppure essere realizzate con piccole schede separate che vengono inserite in opportuni connettori, in modo da realizzare il collegamento fisico con le linee del bus. Inoltre le interfacce sono dotate di connettori per il collegamento dell'unità periferica. Ad esempio, sul retro della scatola del computer sono sempre presenti un connettore per il monitor, uno per la tastiera, uno o due per il collegamento seriale (indicati con COM1 e COM2), e uno per il collegamento parallelo. [10]

[10] Negli ultimi anni si sta diffondendo in modo massiccio l'uso dell'interfaccia

Non mi dilungherò sulla descrizione delle periferiche: vederle anche una sola volta è più efficace di qualunque descrizione sommaria, ed una descrizione tecnica dettagliata delle caratteristiche ci porterebbe troppo lontano. Entrerò nei dettagli più avanti, quando sarà necessario. A questo livello ritengo necessario aggiungere qualche parola solo su nastri e dischi.

Nastri e dischi appartengono alla categoria di periferici denominati "memoria di massa". Si intende con questo termine un supporto di memoria che ha il difetto di non essere immediatamente accessibile alla CPU, ma ha il pregio di poter accumulare una quantità elevata di informazioni e di poterle conservare anche a macchina spenta.

La differenza sostanziale tra dischi e nastri è data dal tipo di accesso alle informazioni. Il nastro ha un tipo di accesso per sua natura sequenziale (sequential access): per accedere ad un'informazione memorizzata a metà del nastro occorre svolgerlo fino al punto che interessa. I dischi hanno un tipo di accesso casuale (random access): le testine di lettura–scrittura possono spostarsi radialmente in modo da raggiungere qualunque posizione del disco. [11]

Con questa premessa vien da domandarsi per quale strana ragione i nastri siano stati frequentemente utilizzati in passato, e qualche esemplare sia sopravvissuto. Ebbene, la ragione c'è, ed è molto semplice: la capacità, soprattutto se considerata in rapporto al costo. Fino a non molti anni fa la quantità di informazioni memorizzabile su un nastro era decisamente superiore o almeno confrontabile con quella memorizzabile anche sui dischi di maggior capacità. Inoltre, il costo di un nastro era decisamente inferiore a quello di un disco rigido. Il nastro era quindi un ottimo supporto per la raccolta di un gran numero di dati, oppure per mantenere copie di riserva dei dati e programmi memorizzati sui dischi — si sa,... i dischi qualche volta si rompono.

Attualmente la situazione è ben diversa: i dischi rigidi hanno ormai un costo inferiore a quello delle unità di lettura–scrittura dei nastri, a fronte di una capacità superiore rispetto a quella delle cartucce. Di conseguenza

---

USB (Universal Serial Bus), che è probabilmente destinata a sostituire in blocco tutte le interfacce con connettori esterni al computer. Il fine – peraltro lodevole – è ridurre il numero di cavi e cavetti vaganti per il collegamento di oggetti vari.

[11] Per capire la differenza pensa a due esempi: il rotolo di pergamena contrapposto al libro, e una cassetta magnetica contrapposta ad un disco musicale. Non è possibile sfogliare un rotolo di pergamena come si fa con le pagine di un libro. Non è possibile inserire una cassetta nel lettore e ascoltare direttamente il brano n. 15 senza aver avvolto il nastro della cassetta fino alla posizione voluta, mentre la stessa operazione era possibile già con i vecchi dischi di vinile, a patto di fare qualche manovra col braccetto che reggeva la puntina, ed lo è oggi in modo decisamente più semplice con gli attuali compact disks — basta sapere qual è il pulsante da premere.

l'uso dei nastri è ormai limitato ad ambienti ove è indispensabile mantenere costantemente copie di riserva di dati e programmi.

Per quanto riguarda i dischi, ve ne sono di diversi tipi.

- Anzitutto – per ragioni storiche – il *floppy disk*.[12] Il basso costo e la facile trasportabilità ne hanno fatto per anni il supporto standard di diffusione di software e dati. I primi esemplari avevano una capacità di memorizzazione di un centinaio di bytes; quelli attualmente più diffusi arrivano normalmente a 1440 Kbytes. Nonostante la capacità ormai decisamente bassa viene ancora montato come standard su tutti i personal computer, perché continua ad essere il punto di partenza per l'installazione di software o per l'intervento in caso di guasto. Alcuni produttori hanno messo sul mercato anche dischi flessibili di diverso tipo che possono arrivare a qualche centinaio di Mbytes.
- Il disco rigido, o *hard disk*. Rappresenta il supporto standard per l'uso quotidiano. Lo sviluppo, in termini di capacità, è stato impressionante: dai 5 Mbytes dei primi anni ottanta si è ormai arrivati a superare la soglia dei 100 Gbytes, e già si intravede all'orizzonte il superamento della soglia del Terabyte (2^{40} bytes...).[13]
- Dischi ottici. Sono nati come supporti a sola lettura, col nome di CD–Rom: i dischi venivano prodotti in fabbrica e distribuiti sul mercato.[14] In seguito, grazie all'immissione sul mercato di masterizzatori a costo sempre più basso, anche gli utenti finali hanno avuto la possibilità di scrivere su dischi CD, ma una sola volta, con una memorizzazione permanente. Attualmente esistono unità in grado di riscrivere più volte anche i dischi ottici (ma attenzione: anche se sembrano uguali a quelli scrivibili una volta sola, il supporto è diverso, e non tutti i masterizzatori sono in grado di trattarli). La capacità dei CD–Rom resta limitata (dai 600 agli 800 Mbytes), soprattutto se confrontata con quella ben superiore dei dischi rigidi, ma il basso costo li ha fatti preferire come

---

[12] Letteralmente: disco floscio, detto così perché il supporto di memorizzazione è in materiale flessibile, protetto una custodia semirigida

[13] Non vorrei essere frainteso: i dischi rigidi non sono nati coi personal computer: esistevano anche molti anni prima, e già alla fine degli anni '70 del secolo scorso c'erano sul mercato dischi di oltre 100 Gbytes. Ma le dimensioni di quelle unità – paragonabili a quelle di una lavatrice – ed il costo elevato non ne avrebbero mai permesso l'inserimento in un personal computer! Tant'è vero che i primissimi cosiddetti "home computer" erano privi di memoria di massa — il software necessario era memorizzato su una memoria ROM, a sola lettura. Successivamente comparvero degli home computer dotati di cassette a nastro magnetico. I primi personal computer simili a quelli attuali montavano solo dischi floppy.

[14] L'acronimo CD-Rom significa appunto "compact disk – read only memory", ovvero disco compatto con memoria a sola lettura.

supporto standard per la distribuzione di software. La versione a maggior capacità, il disco DVD (Digital Versatile Disk), si sta diffondendo rapidamente.

1.2 Uno sguardo dall'alto sul software

Software, come ti ho detto, è, genericamente, l'insieme dei programmi che rendono operante la struttura dell'hardware, nel senso che lo costringono a fare qualcosa. Di utile, si spera. Anche qui, non è possibile descrivere in modo esauriente in poche pagine tutto il software esistente. Ma è possibile, credo, mettere in evidenza alcune linee essenziali che servono almeno per iniziare ad orientarsi.

Una divisione grossolana del software si può fare distinguendo tre categorie:

1. il sistema operativo: un po' come un buon capufficio che ha il compito di controllare il lavoro della macchina;
2. gli strumenti di sviluppo del software: tutto quanto serve per la produzione di nuovo software;
3. i programmi applicativi: quello che interessa all'utente finale, ossia un pacchetto che gli consenta di svolgere un'attività, come comporre un testo, gestire la contabilità domestica o aziendale, provare l'ebbrezza della navigazione – in rete – senza muoversi dalla propria sedia, o divertirsi con uno degli innumerevoli giochi che si trovano in rete o sul mercato.

Nei paragrafi seguenti mi soffermerò un po' più a lungo sul sistema operativo e sullo sviluppo del software. Tralascerò invece la discussione dei pacchetti applicativi: per questo bastano le riviste che puoi trovare a basso costo in qualunque edicola.

1.2.1 Una digressione

Per spiegare a quali esigenze risponda il software un esempio può essere più efficace di qualunque spiegazione astratta. Supponiamo che io mi metta davanti al computer, e decida di comporre il testo che stai leggendo: è proprio ciò che sto facendo in questo momento, che per me è un presente che fluisce continuamente ed irrimediabilmente, e per te, lettore, è ormai un passato remoto. Prova a domandarti: ma cosa ci vuole per scrivere un testo? Uno scrittore di un paio di secoli fa avrebbe risposto, probabilmente: carta e penna, ma soprattutto idee, gusto, un cervello, e l'abilità di sfruttare tutti gli strumenti della lingua per comunicare fatti, sentimenti, emozioni. Per buona parte del secolo scorso lo stesso scrittore avrebbe probabilmente sostituito la penna con una macchina da scrivere, mantenendo invariato tutto il resto. Oggi forse molti direbbero: "un computer!" — dimenticando che servono anche delle idee e la padronanza di una lingua.

Risposta semplice, l'ultima. Ma prova a pensare cosa si nasconde dietro la parola "computer". O meglio, a cosa fai concretamente.

Accendi il computer e avvii un programma di elaborazione testi (ma non ti inquieta un po', questa parola "elaborazione"? Una volta i testi si scrivevano!). Poi batti il testo su una tastiera, guardi quello che hai scritto sullo schermo, lo salvi e infine lo stampi. Facile, no? Dopo tutto con la penna avresti fatto la stessa cosa!...

No, non è la stessa cosa. Lasciami immaginare un dialogo, supponendo che tu abbia già letto per intero il paragrafo sull'hardware.

— *Come hai fatto a scrivere questo testo?*

— Ho acceso il computer; dopo un po' sono comparse delle icone, e ho fatto un clikke–clikke col mouse sull'icona del programma di elaborazione testi.

— *E chi ha fatto comparire le icone?*

— Immagino che ci sia un programma che le disegna sullo schermo.

— *E chi esegue questo programma?*

— La CPU, immagino.

— *E quando tu hai mosso il mouse chi ha spostato il puntatore sullo schermo? Non può averlo fatto il mouse da solo!*

— Ci sarà un programma che quando muovo il mouse sposta il puntatore.

— *Diverso da quello che ha disegnato le icone?*

— Suppongo di sí.

— *E quando hai fatto clikke–clikke cosa è successo?*

— Si è aperta una finestra del programma di elaborazione testi... aspetta: la finestra la deve aver disegnata un programma... diverso dagli altri due.

— *E chi ha fatto partire questo terzo programma?*

— La CPU, naturalmente.

— *E dove è andata a prenderlo?*

— Sul disco, mi sembra ovvio.

— *Ed è andata a prendere sul disco anche gli altri due programmi, quello delle icone e quello del mouse?*

— Ma sí, da dove vuoi che lo prenda?

— *Vabbé, scusa. E poi?*

— Poi ho battuto il testo nella finestra.

— *Un momento: tu il testo l'hai battuto sulla tastiera, e qualcuno l'ha messo nella finestra, immagino.*

— Ma sí, naturalmente ce l'ha messo il programma di elaborazione testi, e l'ha anche impaginato e mi ha corretto gli errori.

— *Pure! Ma come fa il programma di elaborazione testi a trovare gli errori?*

— Bah,... confronterà le parole con quelle che lui conosce. Infatti ogni tanto mi segnala parole che per me sono giuste, ma lui non le sa.

— *Ho capito. Ma come fa a conoscere le parole?*

— Le avrà da qualche parte.

— *Sul disco?*

— E va bene, sul disco!

— *Ce n'è di roba, su quel disco! E poi hai stampato?*

— Sí, ho proprio stampato.

— *Ma chi ha mandato il tuo testo dalla finestra del video alla stampante?*

— Il programma di elaborazione testi, credo.

— *Ma intanto che stampavi hai fatto delle correzioni?*

— Ah, già, ho fatto qualche modifica.

— *E come fa il programma di elaborazione testi a stampare un testo che gli stai modificando sotto il naso?*
— Già!... A dire il vero quando gli ho detto di stampare mi sono accorto che ha fatto qualcosa sul disco. Quindi, forse, ha memorizzato il mio testo da qualche parte e ha detto ad un altro programma di stamparlo. Deve essere andata proprio così.
— *Ma allora i programmi sono diventati quattro?*
— Ehm . . . sí, sono diventati quattro.
— *E sei proprio sicuro che non ce ne siano altri in giro?*
— E chi lo sa?
— *E come fanno quattro programmi a lavorare insieme? La CPU è una sola.*
— Boh? . . .
— *E quando hai fatto le correzioni come gli hai detto che carattere correggere?*
— Mi sono posizionato col mouse, e ho fatto un clikke.
— *E chi si è accorto che hai fatto un clikke?*
— Il programma di elaborazione testi.
— *Ma come ha fatto il programma di elaborazione testi a sapere che il clikke era per lui, e non per il programma delle finestre?*
— Ma che ne so! Sono affari suoi! . . .
(Fine del dialogo. È meglio . . .)

In altre parole, con la potente tecnologia dei calcolatori abbiamo ottenuto lo splendido risultato di trasformare la semplice operazione di prendere una penna e scrivere su un foglio di carta in un guazzabuglio orribile. La tecnologia, si sa, semplifica la vita!

In termini un po' più sintetici, sembra che la situazione sia la seguente: esiste un solo personaggio, la CPU, che fa il lavoro. Per essere in condizioni di farlo deve dare retta a diversi programmi, e gestire diverse periferiche, il tutto senza fare confusione. Ma per poter fare qualcosa la CPU deve eseguire delle istruzioni, cioè un programma. Quindi per gestire dei programmi ci vuole un programma. Ci stiamo avvicinando pericolosamente alla storia dell'uovo e della gallina. Come si fa?

Spero che i paragrafi successivi comincino a fare un po' di luce.

1.2.2 Il sistema operativo

È il gestore di tutte le risorse di sistema, si dice di solito. Ma . . . cosa sono le risorse di sistema? Non è difficile rispondere. L'hardware mette a disposizione:

- una CPU in grado di eseguire programmi di elaborazione dei dati;
- una memoria dove possono essere conservati programmi ed informazioni da elaborare;
- una serie di periferiche;

e a questo si deve aggiungere

- il *tempo*: non è un pezzo di hardware, ma l'elaborazione, così come il trasferimento di dati, non è istantanea: richiede tempo, che, si sa, è prezioso.

Gestire tutto questo è compito del sistema operativo. Si tratta in pratica di un programma che in qualche modo viene attivato al momento dell'accensione della macchina. Non è poi tanto diverso dagli altri programmi, quelli che potresti scrivere tu (e che spero sarai in grado di scrivere alla fine della lettura di queste note). La differenza principale è che deve essere il primo a partire, e quando muore deve provocare l'arresto della macchina — se non vuoi che la macchina si metta a pensare da sola. Per ora, lascia che il come venga attivato resti avvolto nel mistero. Ne riparleremo più avanti.

Il sistema operativo è composto da diversi blocchi:

- un *kernel* (o nucleo) che ne costituisce la parte principale; questa è quella che viene attivata in modo misterioso;
- una serie di *drivers*, ciascuno dei quali si fa carico del controllo di un periferico;
- un *file system* che ha il compito di gestire il contenuto dei dischi.

Il primo personaggio ad essere attivato è il kernel: qualcuno lo porta in memoria e gli dà il via. Tra i suoi compiti principali vi sono:

- l'assegnazione della memoria;
- l'attivazione dei programmi;
- la distribuzione del tempo di elaborazione tra i diversi programmi attivi;
- la sincronizzazione ed il controllo delle operazioni di trasferimento di dati tra la memoria ed i periferici.

Una volta in azione, il kernel provvede immediatamente a crearsi uno schema di gestione della memoria (chi, quale e quanta ne usa). Poi carica in memoria e mette in azione i drivers: uno per la tastiera, uno per il video, uno per il mouse, uno per i dischi, uno per la stampante, &c.

Ciascun driver si assume la responsabilità di coordinare il passaggio di informazioni da e verso un periferico. Per tutti, il referente è il kernel. Se ci sono dei dati da trasferire, ad esempio, sul video, il kernel si rivolge al driver dicendogli: prendi i dati che si trovano nella tal zona di memoria, e trasferiscili alla scheda grafica. Il driver esegue, e torna a dormire. Tradurre quei dati in un'immagine sullo schermo è compito della scheda grafica. Se io muovo il mouse, arriva un segnale all'interfaccia del mouse. Questa avverte il kernel che c'è qualcosa in arrivo, ed il kernel ordina al driver del mouse di andare a vedere, e di mettere i dati che riceve in una certa zona di memoria. Il mouse esegue il trasferimento, e avverte il kernel che l'operazione è finita. È compito del kernel passare i dati al destinatario.

E veniamo all'attivazione dei programmi. Per il momento, lascia perdere le finestre, e supponi di lavorare solo da tastiera. Il kernel lancia il programma di gestione della console — in pratica della tastiera. Se io batto un ordine sulla tastiera, il driver passa al kernel i caratteri che ho battuto, ed il kernel li passa al programma di gestione della tastiera. Se il mio comando ha richiesto l'attivazione di un altro programma – supponiamo che sia quello di elaborazione di testi – il programma di gestione della tastiera passa la richiesta al kernel. Il kernel cerca sul disco il programma che ho richiesto, gli assegna una zona di memoria, lo carica e ne fa partire l'esecuzione. A partire da quel momento, qualunque cosa io batta sulla tastiera viene catturata dal driver, e passata a quel programma (sempre sotto controllo del kernel).

Quando il mio programma termina il kernel lo elimina dalla memoria, e restituisce il controllo della tastiera al programma di gestione della console.

Quello che ho descritto è lo schema più semplice: fondamentalmente quello seguito da un sistema che esegue un solo programma per volta. È molto più interessante avere a disposizione un sistema che gestisca l'esecuzione contemporanea di diversi programmi. Il fatto è che la CPU è in grado di eseguire un solo programma per volta: come si risolve la faccenda?

Ebbene, un programma altro non è che una serie di istruzioni che la CPU deve eseguire in ordine, una alla volta. Se la CPU dedicasse un secondo di tempo ad eseguire un blocco di istruzioni che fanno parte del programma A, poi passasse ad eseguire per un secondo un blocco di istruzioni che fanno parte del programma B, poi tornasse ad A per un secondo &c, non vi sarebbe nulla di male. Così avviene, infatti, almeno sui sistemi un po' evoluti. L'importante è garantire che l'esecuzione di ciascun programma riprenda esattamente dal punto in cui era stata interrotta, senza perdita di informazioni. A questo pensa il kernel.

In termini elementari, la faccenda funziona così. Il registro IP (instruction pointer) della CPU contiene l'indirizzo dell'istruzione da eseguire. Mentre viene eseguito un blocco di programma il registro viene aggiornato di continuo. Un po' come quando, in prima elementare, la maestra ti ha insegnato a leggere tenendo il segno col dito: il registro IP è il dito della CPU. Quando il kernel decide di togliere l'uso della CPU al programma A memorizza da qualche parte il contenuto di IP (e quello degli altri registri generali), e carica in IP l'indirizzo dell'istruzione nuova da eseguire, che fa parte del programma B. Quando toglie il controllo a B per restituirlo ad A salva il contenuto di IP (e dei registri generali), e ripristina quello che c'era nel momento in cui ha tolto il controllo ad A.

Ma, se ci sono tanti programmi attivi, oltre ai drivers, come fa il kernel a decidere a chi concedere il controllo della CPU? Anzitutto, un programma può, con una distinzione grossolana, essere in stato di attesa, oppure essere pronto per l'esecuzione. Considera, ad esempio, il programma di elaborazione testi. Quando io batto un tasto, lui prende il carattere che ho battuto e lo sistema al suo posto. Poi non ha più nulla da fare fin che io non batto il tasto successivo. Il programma comunica al kernel che aspetta dati dalla tastiera, ed il kernel lo mette in stato di attesa. Quindi quel programma non entra in competizione per l'uso della CPU. All'arrivo del carattere successivo il programma rientra nella competizione, ed il kernel gli può assegnare di nuovo il controllo della CPU. Se più programmi sono in competizione il kernel distribuisce il tempo di elaborazione in base a qualche criterio (il più elementare è: un po' per ciascuno). L'attività di distribuzione del tempo ai vari programmi si chiama *scheduling*.

Resta ancora una domanda: per eseguire l'attività di scheduling il kernel deve avere il controllo della CPU. Come fa a toglierlo ad un programma attivo? Usa il meccanismo hardware di *interrupt* (interruzione). Senza entrare in troppi dettagli: la macchina è dotata di un orologio che ad intervalli regolari manda un segnale alla CPU; questa interrompe l'attività in corso e cede il controllo ad un modulo del kernel la cui funzione è, appunto, quella di rispondere al segnale dell'orologio; questo svolge il suo lavoro, e poi passa il controllo al modulo che si occupa dello scheduling. Un meccanismo analogo viene utilizzato per l'attivazione dei drivers: l'interfaccia che è pronta ad inviare o ricevere dati manda un segnale alla CPU tramite il bus di connessione; la CPU interrompe il lavoro in corso per dare il controllo al driver. A trasferimento concluso riprende la normale attività di scheduling.

E veniamo al file system. Dopo aver battuto un testo mi piacerebbe memorizzarlo in modo da poterlo ritrovare, correggere, ristampare &c. Per questo lo devo scrivere su disco, da qualche parte. Ma il disco è semplicemente un piatto rotondo affiancato da testine che possono leggere o scrivere dei dati — ne riparleremo in maggior dettaglio più avanti. Un po' complicato.

È qui che intervengono il driver del disco ed il file system. In termini semplici, la faccenda funziona così. Il driver riordina il contenuto del disco in una una successione di blocchi — come un quaderno con le pagine numerate. Questo rende tutti i dischi praticamente equivalenti in tutto, salvo che per il numero totale di blocchi — ovvero la capacità. Il file system crea un catalogo del tipo: i blocchi di disco da x a y costituiscono un insieme di informazioni – detto *file* (o archivio) – alle quali è assegnato un nome. Quando io ordino al programma di elaborazione testi di memorizzare le note che sto battendo sotto il nome "Appunti" il mio programma si rivolge al file system dicendogli: devi assegnarmi uno spazio di n bytes (la lunghezza del mio testo) per scriverci dei dati; questo insieme di dati lo voglio chiamare "Appunti". Il file system calcola quanti blocchi di disco occorrono arrotondando per eccesso, e cerca nel suo catalogo un gruppo di blocchi liberi. Trovatolo, lo annota come occupato, e scrive nel proprio catalogo che il file di nome "Appunti" occupa i blocchi da x a y del disco. Poi comunica al mio programma che lo spazio è disponibile, ed il mio programma gli risponde: i dati stanno in memoria, al tale indirizzo. Il file system ordina al driver: parti da questo indirizzo di memoria, e trasferisci n bytes ai blocchi da x a y del disco. Il driver traduce i numeri di blocco da x ad y in informazioni che individuano una zona ben determinata del disco e li invia all'interfaccia del disco ordinandole: devi scrivere dei dati in questa zona. L'interfaccia chiede: quali dati? Il driver impacchetta i dati e li trasferisce all'interfaccia. L'interfaccia li invia all'unità disco, che provvede alla scrittura. Semplice, no?

Se più tardi io decido di correggere le mie note devo riattivare il programma di elaborazione testi ed ordinargli di riportare in memoria il contenuto del file "Appunti". Il programma si rivolge al file system ordinandogli:

prendi il contenuto del file “Appunti” e trasferiscilo in memoria a partire dal tal indirizzo. Il file system cerca nel suo catalogo, trova la posizione dei dati sul disco e ordina al driver di procedere al trasferimento.

Se, dopo aver effettuato le correzioni, decido di memorizzare la nuova versione mantenendo la precedente mi basta ordinare al programma di scrivere il mio testo su un file dal nome diverso, ad esempio “Appunti.nuovi” (meglio evitare gli spazi nei nomi dei files). Il file system non sa – e non vuole sapere – che si tratta di un file che ha qualche relazione col precedente: ripete tutta la procedura daccapo. Il risultato è che sul disco ci sono due files, con nomi diversi, che occupano blocchi di disco diversi. Io so che sono due versioni distinte degli stessi appunti, ma il driver ed il file system sono personaggi molto discreti: non ficcano il naso nel contenuto — o forse sono troppo stupidi per farlo.

Se ad un certo punto decido di cancellare la vecchia versione mi basta richiamare un programma – solitamente fornito col sistema – che provvede alla cancellazione, e ordinargli: cancella il file “Appunti”. Il programma si rivolge al file system passandogli l’ordine. Il file system si limita tipicamente ad annotare che i blocchi che erano assegnati al file “Appunti” sono disponibili per una nuova assegnazione, e a cancellare il nome “Appunti” dal suo catalogo. I dati restano, di fatto, sul disco, fin che i blocchi non vengono riassegnati ad un altro file e riscritti.

Un’ultima annotazione. La struttura descritta prevede che i blocchi assegnati ad un file siano consecutivi, o contigui. Può quindi accadere che il file system non trovi lo spazio contiguo per un nuovo file, anche se lo spazio libero sarebbe sufficiente. Questo perché la creazione e cancellazione di files tende a frammentare lo spazio disponibile. Per questo motivo i file system più efficienti prevedono anche la possibilità di memorizzare files su blocchi non contigui, ovviamente al prezzo di una maggiore complessità di gestione delle informazioni.

1.2.3 Lo sviluppo del software

Questo paragrafo cerca di rispondere – sempre in modo sintetico e un po’ rozzo – alla domanda: e come si fa a preparare un programma per un computer? A dire il vero, bisognerebbe prima rispondere ad un’altra domanda: come fa la CPU ad eseguire un programma?

La CPU è un semplice esecutore di istruzioni. Ma cosa sono le istruzioni? Semplice: sono dei codici numerici ai quali corrispondono delle operazioni elementari che la CPU è in grado di eseguire.

Cerco di spiegarmi con un esempio. Prova ad immaginarti la CPU come un uomo chiuso in una stanza e seduto di fronte ad un tavolo; sul tavolo ha un cassettino con l’etichetta IP (il registro Instruction Pointer), una matita, un blocchetto per gli appunti ed un manuale di istruzioni. Lungo la parete della stanza c’è una grande cassettiera, con i cassetti numerati sequenzialmente a partire da 0 (la memoria). Ogni cassetto contiene un foglietto con annotato un numero.

Prova a dare un'occhiata al manuale di istruzioni. C'è scritto qualcosa del tipo:

0 `HALT`: mettiti a riposo

1 `CLEAR`:
- ○ prendi il numero che si trova nel registro IP, e annotalo come a;
- ○ apri il cassetto di memoria col numero a;
- ○ scrivi 0 sul foglietto che c'è nel cassetto;
- ○ richiudi il cassetto e torna al tuo tavolo.

2 `STORE`:
- ○ prendi il numero che si trova nel registro IP, e annotalo come a;
- ○ apri il cassetto di memoria col numero a;
- ○ leggi il numero che si trova nel cassetto ed annotalo come c;
- ○ prendi il numero che si trova nel registro IP, e annotalo come b;
- ○ apri il cassetto di memoria col numero b;
- ○ leggi il numero che si trova nel cassetto b ed annotalo come d;
- ○ scrivi il numero c sul foglietto che c'è nel cassetto d;
- ○ richiudi tutti i cassetti che hai aperto e torna al tuo tavolo.

3 `MOVE`:
- ○ prendi il numero che si trova nel registro IP, e annotalo come a;
- ○ apri il cassetto di memoria col numero a;
- ○ leggi il numero che si trova nel cassetto ed annotalo come c;
- ○ apri il cassetto di memoria col numero c;
- ○ leggi il numero che si trova nel cassetto c ed annotalo come d;
- ○ prendi il numero che si trova nel registro IP, e annotalo come b;
- ○ apri il cassetto di memoria col numero b;
- ○ leggi il numero che si trova nel cassetto b ed annotalo come e;
- ○ scrivi il numero d sul foglietto che c'è nel cassetto e;
- ○ richiudi tutti i cassetti che hai aperto e torna al tuo tavolo.

4 ...

Mentre ti stai lambiccando il cervello per immaginare a cosa serva questa robaccia ti accorgi che nel manuale c'è anche un'introduzione (quella che nessuno legge mai) che dice:

Istruzioni generali:

(i) leggi il numero contenuto nel registro IP, e annotalo come x;
(ii) apri il cassetto di memoria col numero x;
(iii) leggi il numero che si trova nel cassetto ed annotalo come y;
(iv) cerca sul manuale il numero di istruzione y;
(v) esegui l'istruzione;
(vi) torna al passo (i).

IMPORTANTE: ogni volta che leggi il numero del foglietto che si trova nel registro IP devi incrementarlo di 1 e rimettere il foglietto nel registro IP.

Beh, qui si comincia a capire, . . . forse. Le istruzioni generali spiegano all’uomo della stanza quale debba essere la sua attività: deve semplicemente ripetere ciclicamente le operazioni indicate nelle istruzioni generali. La sola variante è al passo (v): l’operazione da eseguire dipende dal codice che ha trovato, ma è spiegata in dettaglio nel resto del manuale. L’elenco delle istruzioni è formato da una successione di numeri; a ciascun numero corrisponde un codice mnemonico ed un elenco dettagliato di operazioni da compiere. L’uomo non è autorizzato a pensare: deve solo eseguire.

Supponi ora che il registro IP contenga il numero 43, e che i cassetti di memoria a partire dal numero 43 contengano, in successione, i numeri 1, 78, 2, 10, 133, 3, 181, 187, 0. Cosa ci sia negli altri cassetti non è molto rilevante. Immagina cosa farà l’uomo della stanza. Qui sotto trovi la spiegazione: la prima colonna è il numero del cassetto, la seconda il contenuto, la terza è la serie di operazioni eseguite dall’uomo della stanza. Il simbolo # indica la fine di un’istruzione.

43	1	preleva il numero 43 da IP, lo incrementa e mette il numero 44 in IP; apre il cassetto di memoria 43 e ci trova il numero 1: è l’istruzione `CLEAR`;
44	78	preleva il numero 44 da IP; mette 45 in IP; estrae il numero 78 dal cassetto 44; azzera il contenuto del cassetto 78; #
45	2	preleva il numero 45 da IP; mette 46 in IP; apre il cassetto di memoria 45 e ci trova il numero 2: è l’istruzione `STORE`;
46	10	preleva il numero 46 da IP; mette 47 in IP; estrae il numero 10 dal cassetto 46;
47	133	preleva il numero 47 da IP; mette 48 in IP; estrae il numero 133 dal cassetto 47; deposita il numero 10 nel cassetto 133; #
48	3	preleva il numero 48 da IP; mette 49 in IP; estrae il numero 3 dal cassetto 48: è l’istruzione `MOVE`
49	181	preleva il numero 49 da IP; mette 50 in IP; estrae il numero 181 dal cassetto 49;
50	187	preleva il numero 50 da IP; mette 51 in IP; estrae il numero 187 dal cassetto 50; copia il contenuto del cassetto 181 nel cassetto 187; #
51	0	preleva il numero 51 da IP; mette 52 in IP; estrae il numero 0 dal cassetto 48: è l’istruzione `HALT`; stanco, ma soddisfatto, esegue: (*bzzzz . . . ron . . .*) #

Il programma è finito. Non che abbia fatto granché, ma l’obiettivo era solo illustrare come lavora l’uomo della stanza. Un altro esempio di come una cosa semplice possa essere resa complicata dalla tecnologia. Ed anche per illustrare un fatto da tener bene a mente: il calcolatore non è intelligente, è solo veloce — una faccenda che qualcuno dovrebbe spiegare a certi ministri che vorrebbero riformare la scuola sostituendo i maestri ed i libri coi computer.

Torniamo all’esposizione generale. Un programma, nella forma che la

CPU possa utilizzare, è soltanto una successione di istruzioni che vengono scritte in memoria. Un'istruzione è formata da un codice numerico che specifica cosa la CPU debba fare, seguito da uno o più numeri (quanti, dipende dall'istruzione) che specificano gli indirizzi di memoria su cui agire. I codici di istruzioni comprensibili alla CPU sono quelli previsti da chi l'ha progettata e costruita, e sono descritti nel manuale della CPU stessa.

Va da sé che chi vuol scrivere un programma deve preparare tutta la successione di numeri che costituiscono il programma stesso. Un lavoro immane, ma i primissimi calcolatori si programmavano proprio così! Fortunatamente, adesso le cose vanno meglio. [15]

Il primo passo per semplificare la programmazione è il linguaggio *Assembler*. Nella sua forma più semplice consiste nel sostituire ai codici numerici delle istruzioni dei codici mnemonici. Ad esempio, il programmino che ho illustrato sopra potrebbe essere scritto così (tutto quello che segue il carattere punto e virgola è commento che serve a te, ma per la macchina non ha nessun significato):

```
CLEAR 78        ; azzera il contenuto del cassetto 78
STORE 10,133    ; metti il numero 10 nel cassetto 133
MOVE  181,187   ; copia dal cassetto 181 al cassetto 187
HALT            ; mettiti a dormire
```

Decisamente più facile da ricordare. Peccato che la macchina non lo capisca direttamente: occorre un traduttore che sostituisca i codici mnemonici con quelli numerici. Questo lo svolge un programma (un altro!...): il traduttore del linguaggio Assembler. Anzi, questo traduttore sa fare anche di meglio: ti permette di indicare gli indirizzi di memoria con dei nomi, anziché inserire direttamente dei numeri, e di sostituire le sequenze di istruzioni usate più frequentemente con dei nomi simbolici, detti *Macro istruzioni* o più brevemente *Macro*, che il traduttore stesso espande in istruzioni elementari e poi converte nella forma definitiva di codici numerici.

Naturalmente, anche l'Assembler come linguaggio non ha molto di naturale — non capita molto spesso di sentir qualcuno ordinare un caffè in un bar parlando in assembler. E, ancor più grave, ogni modello di CPU ha il suo insieme di istruzioni, il che rende difficile trasferire un programma tra CPU diverse senza modifiche. Per questi motivi sono stati sviluppati linguaggi più evoluti, idealmente indipendenti dalla macchina: BASIC, FORTRAN, COBOL, C, ALGOL, LISP, PL-I, PASCAL, LISP, RPG, C++, JAVA per citarne solo alcuni (la mia ignoranza non mi consente di elencarli tutti). Ciascuno ha il suo campo specifico di applicazione. Ad esempio, il linguaggio C

---

[15] Per essere precisi, il processo di preparazione di un programma è diventato ancora più complesso, ma noi non ce ne accorgiamo perché il grosso del lavoro di routine – quello che richiede scarsa immaginazione ma molta precisione – lo fa la macchina.

è usato per sviluppare il sistema operativo UNIX e tutti suoi cloni — ormai ampiamente diffusi; il linguaggio FORTRAN è stato sviluppato per applicazioni scientifiche, ma è stato talvolta usato per farci di tutto, compresa la stampa delle fatture; il linguaggio COBOL ha avuto una diffusione molto ampia per le applicazioni commerciali, e così via. A ciascun linguaggio corrisponde un programma — detto *compilatore*, che ha il compito di tradurre le istruzioni del linguaggio nei codici numerici interpretabili dalla CPU. [16]

Descrivere in maggior dettaglio i linguaggi non è mia intenzione — e probabilmente non sarei neppure in grado di farlo in modo soddisfacente. C'è però un aspetto comune a tutti i linguaggi, ed è quello che traspare dall'esempio discusso fin qui: un programma è e resta una successione di istruzioni scelte in un insieme predefinito, che la macchina esegue in modo sequenziale, senza fare salti non esplicitamente programmati e senza prendere iniziative personali. Dunque, scrivere un programma significa tradurre un progetto di lavoro in una successione di operazioni elementari che il linguaggio sappia descrivere.

[16] Probabilmente ti è venuta spontanea una domanda: e chi scrive il compilatore che linguaggio usa? La risposta è nascosta nel testo. Se sono in grado di scrivere un programma elementare in linguaggio binario posso ben scriverne uno che traduce le istruzioni mnemoniche in codici numerici. Poi, usando le istruzioni mnemoniche, ne scrivo uno che traduce anche delle macro istruzioni, poi... continuando di questo passo arrivo alla conclusione che il compilatore del linguaggio C è scritto... in linguaggio C. E così è, infatti: una versione del compilatore serve per preparare la successiva.