Corso di Laurea in Ingegneria Informatica e delle Telecomunicazioni
Corso di Fondamenti di Informatica

Estratto ad uso didattico dal testo "Informatica: arte e mestiere"
di S. Ceri, D. Mandrioli, L. Sbattella
McGraw-Hill Libri Italia srl, 1999.

2

Architettura di un calcolatore

Nel Capitolo 1 abbiamo presentato una sommaria introduzione all'informatica, con particolare attenzione al concetto di algoritmo; in questo capitolo introduciamo invece i concetti fondamentali che sono alla base dell'architettura di un calcolatore. Queste due nozioni – l'algoritmo e l'architettura dei calcolatori – sono da ritenersi propedeutiche alla programmazione, argomento illustrato nella Parte prima.

L'architettura della maggior parte degli elaboratori elettronici è organizzata secondo il modello della cosiddetta **macchina di von Neumann**, dal nome del ricercatore americano che, nel corso della Seconda Guerra Mondiale, dedicò i suoi studi alla realizzazione dei primi elaboratori; una delle prime applicazioni degli elaboratori fu la codifica e decodifica dei messaggi scambiati all'interno dell'esercito americano.

Scopo di questo capitolo è di far comprendere al lettore come non accada nulla di "magico" quando un calcolatore esegue un programma (cioè un algoritmo codificato), utilizzando la macchina di von Neumann come esempio illustrativo.

Il Paragrafo 2.1 spiega quali sono i componenti principali dell'architettura di von Neumann. Il Paragrafo 2.2 descrive mediante alcuni esempi come vengono rappresentati, entro la macchina, i dati e le istruzioni che costituiscono un programma per avere una corretta esecuzione del programma stesso. Nei paragrafi successivi viene analizzato il comportamento di ogni unità funzionale dell'architettura di von Neumann e viene mostrato il funzionamento complessivo della macchina in riferimento all'esecuzione di un semplice programma. Una trattazione approfondita della codifica binaria dei dati e delle istruzioni, nonché di struttura e comportamento di architetture tradizionali e avanzate, viene fornita dai Capitoli 11 e 12.

2.1 Elementi della macchina di von Neumann

La macchina di von Neumann è costituita da quattro elementi funzionali fondamentali: l'*unità di elaborazione* (CPU, *Central Processing Unit*), la *memoria centrale*, le *periferiche* e il *bus di sistema*. Questi componenti sono stati brevemente illustrati nel Paragrafo 1.4.1; la Figura 2.1 mostra il modo in cui interagiscono.

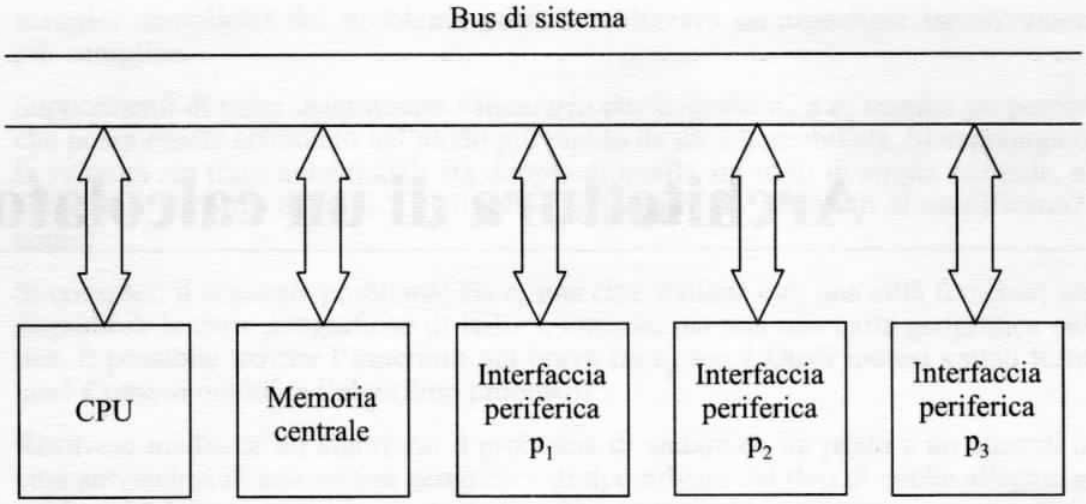

Figura 2.1 Architettura della macchina di von Neumann

L'**unità di elaborazione** contiene i dispositivi elettronici in grado di acquisire, interpretare ed eseguire le istruzioni del programma. La **memoria centrale** contiene le informazioni necessarie all'esecuzione di un programma, cioè istruzioni e dati. Le **periferiche** permettono lo scambio delle informazioni fra l'elaboratore e il mondo esterno, attraverso operazioni di ingresso – dall'esterno all'elaboratore – e di uscita – dall'elaboratore al mondo esterno –. In particolare, fanno parte del calcolatore le sole **interfacce** di collegamento verso le periferiche, mentre queste ultime vengono considerate come dispositivi separati. Nell'architettura di von Neumann, le periferiche includono anche le **memorie di massa**, che dal punto di vista dell'interazione con il calcolatore si comportano in modo funzionalmente analogo a terminali, stampanti e altri dispositivi collegati verso l'esterno. Infine, il **bus di sistema** opera il collegamento fra questi elementi funzionali. Questa architettura elementare è un'astrazione della macchina reale, in cui possono essere presenti componenti ulteriori; tuttavia, tale astrazione è sufficientemente vicina alla realtà da dare una buona idea degli elementi funzionali di un calcolatore.

Il funzionamento della macchina di von Neumann può essere schematizzato come segue. L'unità di elaborazione coordina le varie attività, in particolare **estrae** istruzioni dalla memoria, le **decodifica** comprendendo il loro significato e le **esegue** tramite opportune funzioni all'interno dell'architettura del calcolatore. Le istruzioni possono comportare operazioni di *elaborazione* dell'informazione (per esempio, operazioni numeriche) oppure operazioni di *trasferimento* dell'informazione (per esempio, di dati dall'interfaccia di una periferica alla memoria centrale).

I trasferimenti tra elementi funzionali diversi avvengono sempre tramite il bus di sistema che, in funzione dell'operazione in atto, effettua il collegamento logico tra i vari elementi funzionali coinvolti nel trasferimento (mentre il collegamento fisico è sempre presente). Nella macchina di von Neumann le fasi di elaborazione si succedono in modo sincrono rispetto alla scansione temporale imposta da un **orologio (clock) di sistema**. Durante ciascun intervallo di tempo una parte dell'unità di elaborazione (detta **unità di controllo**) coordina l'esecuzione temporale delle funzioni

che verranno svolte internamente all'unità di elaborazione stessa o negli altri elementi funzionali.

Il modello di von Neumann, anche se ha ormai quasi cinquant'anni di vita, è tuttora adottato dalla maggior parte dei calcolatori. Il suo principale limite è che tutte le operazioni vengono eseguite in stretta sequenza, determinata dall'unità di elaborazione.

Modelli evoluti di questa architettura prevedono di introdurre varie forme di **parallelismo**, cioè di esecuzione contemporanea delle attività di elaborazione.

2.2 Codifica dei dati e delle istruzioni di programma

In un calcolatore, i dati e le istruzioni di programma sono codificate in forma **binaria**, ossia in una sequenza finita di `1` e di `0`. La più piccola unità di informazione memorizzabile o elaborabile da un calcolatore, il **bit** (forma contratta per *binary digit*), corrisponde allo stato di un dispositivo fisico che viene interpretato come `1` o come `0`. Un'altra importante unità di informazione è il **byte**, pari a otto bit. Un byte può essere usato per produrre 2^8 differenti sequenze di `1` e di `0` (`00000000`, `00000001`, `00000010`, ..., `11111111`).

Un calcolatore può trattare diversi tipi di dati: numeri (naturali, interi, reali, frazionari), testi, immagini, suoni ecc. Tutti i dati, non importa quanto complessi siano, devono essere trasformati in sequenze di bit per poter essere elaborati. Esistono diverse convenzioni per codificare i dati di cui sopra. Non intendiamo dare a questo punto una descrizione completa della rappresentazione binaria dell'informazione; preferiamo rimandare questo argomento al Capitolo 11. Tuttavia mostreremo, per mezzo di esempi, che tale rappresentazione binaria è possibile.

- Otto bit sono sufficienti per rappresentare i *numeri naturali* da 0 a 255 (= $2^8 - 1$). Lo zero viene rappresentato in otto bit dalla sequenza `00000000`; il numero 8 da `00001000`; il 255 da `11111111`.
- I *numeri interi* comprendono i numeri interi negativi, lo zero e i numeri interi positivi, per cui va rappresentato anche il loro segno. In pratica, il primo bit del byte viene usato per l'attribuzione del segno; per convenzione, uno `0` indica un numero positivo, mentre un `1` indica un numero negativo. Pertanto, otto bit possono rappresentare gli interi compresi fra –127 e 127, ossia fra $-(2^{(8-1)} - 1)$ e $(2^{(8-1)} - 1)$.
- I *numeri reali*, come è noto, sono numeri razionali contenenti una parte intera e una frazionaria che approssima il numero reale con precisione arbitraria. Se viene usata la *notazione in virgola fissa*, un numero reale può essere rappresentato codificando separatamente la sua parte intera e la sua parte frazionaria. In questo caso, per esempio, il numero 8.345 sarebbe rappresentato in base due come segue:

  primo byte (rappresentazione dell'intero 8) = `00001000`
  secondo byte (rappresentazione della parte frazionaria 0.345) = `01011000`

- I *caratteri* che costituiscono un testo sono codificati in sequenze di bit mediante un codice di traduzione. Quello più diffuso è il codice ASCII (*American Standard Code for Information Interchange*); esso impiega sette bit, per cui permette la rappresentazione di un massimo di 128 caratteri. Il codice ASCII assegna a ogni lettera (le maiuscole da `A` a `Z`, le minuscole da `a` a `z`), cifra (da `0` a `9`) o separatore (usato per la punteggiatura o come operatore aritmetico) un numero naturale rappresentabile in forma binaria. Per esempio, la lettera "A" viene codificata in ASCII come numero 65 e la sua forma binaria è `01000001`; il separatore ";" viene codificato come 59 e la sua forma binaria è `00111011`.

Come abbiamo detto in precedenza, le istruzioni nel cosiddetto codice oggetto (eseguibile) possono denotare operazioni per l'elaborazione delle informazioni (per esempio, operazioni numeriche) od operazioni per il trasferimento delle informazioni da un elemento della macchina a un altro. Un'istruzione codificata presenta in genere due parti, un codice operativo e uno o più operandi. Il **codice operativo** specifica l'operazione da compiere; gli **operandi** specificano in vari modi dove la macchina può reperire i dati coinvolti nella elaborazione o nel trasferimento. Il codice operativo e gli operandi sono codificati in forma binaria usando particolari convenzioni.

2.3 Comportamento della macchina di von Neumann

Questo paragrafo descrive il comportamento di ciascuna unità funzionale della architettura di von Neumann mostrata in Figura 2.1. Particolare attenzione viene prestata al ruolo centrale del processore e allo scambio di informazioni fra le differenti unità coinvolte nell'esecuzione dei programmi.

2.3.1 La memoria centrale

La memoria centrale è destinata ad accogliere il "materiale di lavoro" su cui l'elaboratore opera: dati e programmi, cioè sequenze di istruzioni. Se vogliamo fare un'analogia con l'organizzazione della mente umana, la memoria centrale contiene informazione a breve o medio termine, mentre nella memoria di massa viene memorizzata informazione a lungo termine. La memoria centrale è in generale di dimensioni ridotte e può quindi accogliere solo una parte dell'informazione disponibile. Tuttavia, essa è un "passaggio obbligato": prima di poter essere elaborata, l'informazione deve essere acquisita dalla memoria centrale. In genere, ciò comporta una operazione di ingresso/uscita, in cui cioè l'informazione viene trasferita dalla memoria di massa a quella centrale o viceversa. Discuteremo le caratteristiche di queste operazioni nel Capitolo 13.

Da un punto di vista concettuale, la memoria centrale è una sequenza di celle di memoria; ciascuna cella contiene una *parola* (*word*). Le **parole** sono, quindi, sequenze di bit; ciascuna parola in memoria assumerà durante la computazione un particolare valore (per esempio, con parole di 16 bit, il numero 4327 o la stringa di caratteri "mi") in base alla sequenza di `0` e `1` che verrà inserita nella corrispondente

cella di memoria. Le celle di memoria di un elaboratore hanno tutte la stessa capacità (e, di conseguenza, le parole di uno stesso elaboratore hanno tutte la stessa lunghezza). Viceversa, elaboratori differenti possono avere parole di lunghezza differente. Le più tipiche lunghezze di parola includono i primi multipli del byte (8 bit), e si hanno perciò calcolatori con parole di 8, 16, 32 e 64 bit.

Tecnologicamente, le memorie sono realizzate con dispositivi a semiconduttori; una memoria può essere idealizzata come una grossa "tabella", che ha per righe le varie celle; le colonne, in numero pari alla lunghezza di parola, individuano ciascun bit di memoria. L'informazione è presente in memoria come stato (alto o basso) di tensione nelle posizioni della memoria individuate agli incroci delle righe con le colonne. Assumiamo che il valore basso di tensione sia associato al valore `0` e il valore alto sia associato al valore `1`. La Figura 2.2 descrive schematicamente la struttura della memoria.

In genere, la memoria centrale è *volatile*, cioè il suo contenuto viene perduto quando il calcolatore viene spento (o quando, per esempio, viene a mancare l'energia elettrica); nel riprendere dopo una interruzione, il valore degli `0` e `1` nelle celle di memoria non è significativo. Viceversa, la memoria di massa è *permanente*, e quindi l'informazione in essa contenuta non va persa quando il calcolatore viene spento. Tuttavia, alcune memorie centrali di nuova concezione sono alimentate da batterie autonome e divengono pertanto anch'esse permanenti; ciò consente di migliorare la "robustezza" di alcune applicazioni informatiche, rendendole meno soggette a guasti o malfunzionamenti.

Ciascuna cella di memoria può essere *indirizzata*; con questo termine si indica la capacità dell'elaboratore di selezionare una particolare cella di memoria. L'**indirizzo** di una cella di memoria è semplicemente la sua posizione relativa (o numero d'ordine) rispetto alla prima cella di memoria, cui viene normalmente attribuita la posizione zero. L'indirizzamento della memoria avviene tramite un opportuno registro, detto **registro indirizzi**, che si trova nell'unità di elaborazione. In generale, un registro è un dispositivo elettronico capace di memorizzare una sequenza di bit. Come abbiamo visto nel paragrafo precedente, sequenze di bit possono venire interpretate come numeri; in particolare, i bit contenuti nei registri vengono interpretati

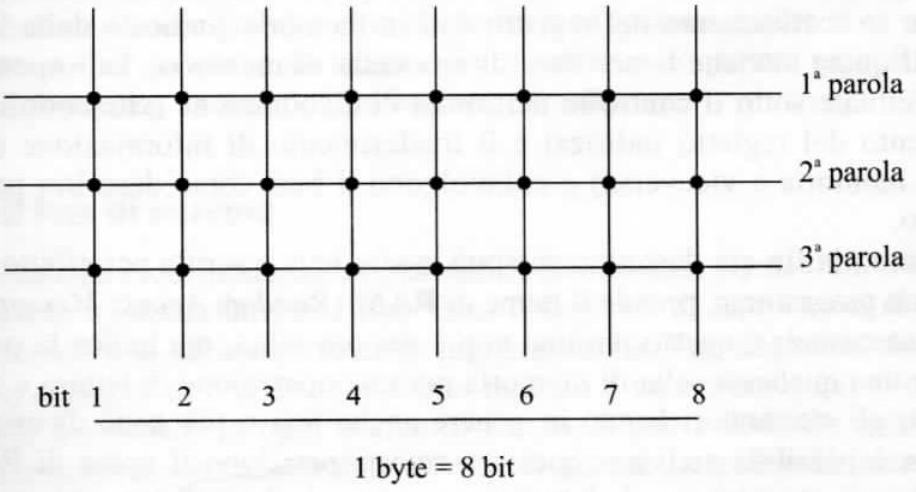

Figura 2.2 Struttura della memoria

come numeri naturali. Se il registro indirizzi ha k bit, si possono indirizzare fino a 2^k celle di memoria, i cui indirizzi varieranno fra 0 e $2^k - 1$.

Questo meccanismo indica il motivo per cui anche la dimensione della memoria, cioè il numero di parole disponibili, è in genere una potenza di 2. In particolare, se il registro indirizzi è lungo 10 bit vengono indirizzate $2^{10} = 1024$ celle; questa dimensione viene indicata come "kilo-parola" (più in generale, il termine "kilo" viene in genere associato alla potenza 2^{10}, contrariamente al significato usuale). Se invece il registro indirizzi è lungo 20 bit vengono indirizzate $2^{20} = 1048576$ celle; si parla in tal caso di "mega-parola" (il termine "mega" viene associato alla potenza 2^{20}). Si noti però che la capacità della memoria centrale viene normalmente misurata indicando il numero complessivo di byte disponibili (e quindi il numero complessivo di megabyte o di gigabyte) e non in numero di parole, perché la dimensione in byte è indipendente dalla lunghezza di parola di uno specifico calcolatore, e quindi è una unità di misura più generale.

È dunque possibile selezionare una specifica cella di memoria caricando il registro indirizzi con una sequenza di `0` e `1`, che indichi la posizione relativa della cella nella memoria. A questo punto, si possono effettuare due operazioni: la *lettura dalla memoria* e la *scrittura in memoria*.

Entrambe le operazioni utilizzano un secondo registro dell'unità di elaborazione, detto **registro dati**, che è lungo come una parola di memoria. L'operazione di lettura provoca la copia del contenuto della cella di memoria nel registro dati; essa **carica** (in inglese, *load*) il registro dati con una parola di memoria. L'operazione di scrittura copia il contenuto del registro dati in una cella di memoria; essa **deposita** (in inglese, *store*) il contenuto del registro dati in una parola di memoria. Queste operazioni possono essere visualizzate pensando che i valori (`0` e `1`) vengano copiati dalla cella indirizzata al registro dati o viceversa. In realtà, i circuiti elettrici della memoria sono opportunamente collegati fra loro in modo tale che i valori di tensione (alta o bassa) vengano ordinatamente trasmessi.

Il comportamento della memoria è riassunto nella Figura 2.3. Si noti il registro indirizzi (lungo k bit), la memoria (con 2^k celle, ciascuna dotata di parole di h bit), e il registro dati (lungo h bit). Le operazioni disponibili sono: il caricamento del registro dati (indicato dalla lettera "L"), tramite il quale avviene la lettura di una cella di memoria; lo scaricamento del registro dati in memoria (indicato dalla lettera "S"), tramite il quale avviene la scrittura di una cella di memoria. Tali operazioni vengono effettuate sotto il controllo dell'unità di elaborazione (che coordina anche il caricamento del registro indirizzi e il trasferimento di informazione dal registro dati alla memoria e viceversa) e coinvolgono il bus, come descritto nel prossimo paragrafo.

La memoria fin qui descritta, che può essere letta e scritta per effetto delle istruzioni di un programma, prende il nome di **RAM** (*Random Access Memory*, memoria ad accesso casuale); questo termine non è dei più felici, ma indica la possibilità di scegliere una qualsiasi cella di memoria per una operazione di lettura e scrittura. In contrasto, gli elaboratori hanno in genere anche una o più zone di memoria sulle quali non è possibile scrivere; queste memorie prendono il nome di **ROM** (*Read Only Memory*, memoria a sola lettura); esse vengono inizializzate dal costruttore del calcolatore con dati e programmi che servono a far funzionare il sistema.

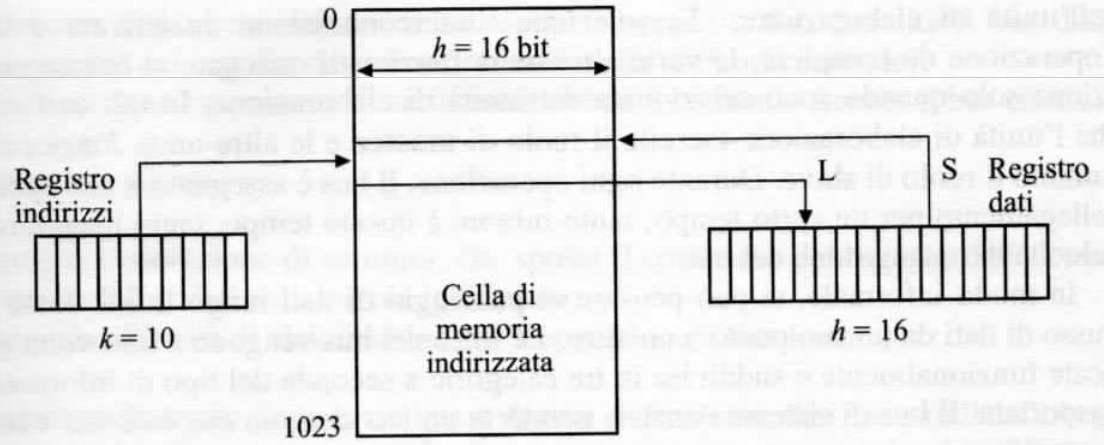

Figura 2.3 Schema di funzionamento della memoria

Il fatto di poter solo leggere le memorie ROM può sembrare a prima vista una limitazione; in realtà, in questo modo il contenuto delle memorie ROM viene protetto, cioè viene preservato da guasti o scritture operate da programmi scorretti. Inoltre, il contenuto della memoria ROM è persistente. Per queste caratteristiche, le memorie ROM sono usate nei microprocessori presenti in dispositivi di controllo di vario genere, per esempio nelle lavastoviglie o nei sistemi istallati a bordo delle automobili o degli aerei.

Le memorie ROM si distinguono in varie categorie a seconda del modo in cui vengono scritte dai costruttori di elaboratori. Alcune di esse, dette EROM (*Erasable-ROM*, cioè ROM cancellabili) possono essere cancellate, dopo essere state estratte dal calcolatore, sottoponendole a raggi ultravioletti. In tal caso, la stessa memoria fisica può essere riutilizzata più volte, anche se tramite un processo laborioso. La scrittura della memoria può avvenire durante il processo di costruzione, oppure essere effettuata da particolari dispositivi, detti programmatori di ROM; in tal caso, si parla di PROM (*Programmable ROM*, cioè ROM programmabili). La combinazione di programmabilità e cancellabilità corrisponde è offerta dalle memorie EPROM (*Erasable Programmable ROM*). Questa descrizione ci fa capire che le memorie ROM hanno alcune caratteristiche dell'hardware (non possono venire modificate durante l'esecuzione) e altre del software (possono essere programmate); in realtà, proprio per questo ruolo intermedio, il software contenuto nelle ROM prende il nome di *firmware*, "a cavallo" fra hardware e software.

2.3.2 Il bus di sistema

Il bus di sistema è costituito da un insieme di connessioni elementari, o linee, lungo le quali viene trasferita informazione. Esso collega fra di loro l'unità di elaborazione, la memoria e le varie interfacce di ingresso/uscita. Topologicamente, il bus è in genere un collegamento aperto (cioè non limitato a un estremo) cui si collegano le varie unità funzionali. In ogni istante di tempo, il bus è dedicato a collegare due unità funzionali: una trasmette dati e l'altra li riceve. Le possibili interconnessioni sono tra l'unità di elaborazione e la memoria, oppure tra l'unità di elaborazione e l'interfaccia di una specifica periferica. Il bus è in genere sotto il controllo

dell'unità di elaborazione, che seleziona l'interconnessione da attivare e indica l'operazione da compiere; le varie altre unità funzionali collegate al bus entrano in azione solo quando sono selezionate dall'unità di elaborazione. In tali casi si dice che l'unità di elaborazione esercita il ruolo di **master** e le altre unità funzionali assumono il ruolo di **slave**. Durante ogni operazione, il bus è assegnato a uno specifico collegamento per un certo tempo; tanto minore è questo tempo, tanto maggiore è la velocità di trasmissione del bus.

In modo informale, si può pensare al passaggio di dati lungo il bus come a un flusso di dati da un suo punto a un altro. Le linee del bus vengono a loro volta specificate funzionalmente e suddivise in tre categorie a seconda del tipo di informazione trasportata. Il bus di sistema consiste perciò in un *bus dati*, un *bus indirizzi* e un *bus controlli*.

- Il **bus dati** trasferisce dati dall'unità master all'unità slave o viceversa. Se per esempio l'unità slave è la memoria centrale, i dati vengono trasferiti da una cella di memoria al registro dati a seguito di una operazione di lettura, oppure dal registro dati a una cella di memoria a seguito di una operazione di scrittura.
- Il **bus indirizzi** serve, per esempio, per trasmettere il contenuto del registro indirizzi alla memoria centrale; in questo modo, viene selezionata una specifica cella di memoria per una operazione di lettura o scrittura. Perché questa operazione sia possibile, il trasferimento dell'informazione avviene dall'unità di elaborazione alla memoria.
- Il **bus controlli** trasferisce dall'unità master all'unità slave un codice corrispondente all'istruzione da eseguire e dall'unità slave all'unità master informazioni relative all'avvenuto espletamento dell'operazione richiesta.

Per effettuare un'operazione di *lettura* dalla memoria centrale, l'unità di elaborazione deve pertanto:

1. caricare l'indirizzo della parola di memoria che desidera leggere nel registro indirizzi e trasmetterlo alla memoria centrale tramite il bus indirizzi;
2. richiedere un'operazione di lettura inviando il comando tramite il bus controlli.

La memoria allora:

3. esegue l'operazione di lettura, che sposta tramite il bus dati il contenuto della parola indirizzata nel registro dati;
4. segnala all'unità di elaborazione, tramite il bus controlli, che l'operazione è terminata e che il dato richiesto è disponibile nel registro dati.

Per effettuare un'operazione di *scrittura* in memoria centrale, l'unità di elaborazione deve invece:

1. caricare l'indirizzo della parola di memoria in cui desidera scrivere nel registro indirizzi e trasmetterlo alla memoria centrale tramite il bus indirizzi;

2. caricare il dato da scrivere in memoria nel registro dati (tramite il bus dati il contenuto del registro dati viene trasmesso alla memoria centrale);
3. richiedere un'operazione di scrittura inviando il comando tramite il bus controlli.

La memoria allora:

4. esegue l'operazione di scrittura, che sposta il contenuto del registro dati ricevuto tramite il bus dati nella parola indirizzata;
5. segnala all'unità di elaborazione che l'operazione è terminata.

La presenza di molte linee in un bus consente di trasferire dati "in parallelo"; in genere, tutti i bit di una stessa parola di memoria vengono trasmessi sul bus dati allo stesso istante, dedicando una linea a ciascuno di essi. Alternativamente, la trasmissione può richiedere più di un trasferimento in corrispondenza di una stessa parola di memoria; ciò è necessario quando il numero di linee del bus è inferiore alla lunghezza delle parole di memoria.

2.3.3 L'unità di elaborazione

L'unità di elaborazione (CPU) contiene gli elementi circuitali che regolano il funzionamento dell'elaboratore. La sua funzione è quella di eseguire i programmi contenuti nella memoria centrale prelevando, decodificando ed eseguendo una dopo l'altra le istruzioni che li costituiscono. Gli elementi circuitali che costituiscono la CPU sono illustrati in Figura 2.4:

- l'**unità di controllo** è responsabile del prelievo e della decodifica delle istruzioni nonché dell'invio dei segnali di controllo che provocano i trasferimenti o le elaborazioni necessarie per l'esecuzione dell'istruzione decodificata;
- l'**orologio di sistema** (*clock*) sincronizza le operazioni rispetto a una data frequenza;
- l'**unità aritmetico-logica** (ALU, *Arithmetic and Logic Unit*) realizza le operazioni aritmetiche e logiche eventualmente richieste per l'esecuzione dell'istruzione (come le operazioni aritmetiche che devono essere eseguite sui dati quando richiesto dal programma).

Inoltre, la CPU ha diversi registri. Come abbiamo già detto, un **registro** è un elemento di memoria che può essere letto o scritto molto velocemente, e che è utilizzabile per memorizzare risultati parziali o informazioni necessarie al controllo. I principali registri dell'unità di elaborazione sono:

- il **registro dati** (DR, *Data Register*) e il **registro indirizzi** (AR, *Address Register*), già descritti nel Paragrafo 2.3.1. Ricordiamo che AR è lungo k bit e DR è lungo h bit, ossia quanto la lunghezza di una parola;
- il **registro istruzione corrente** (CIR, *Current Instruction Register*), lungo h bit, che contiene, istante per istante, l'istruzione che risulta in esecuzione da parte dell'elaboratore;

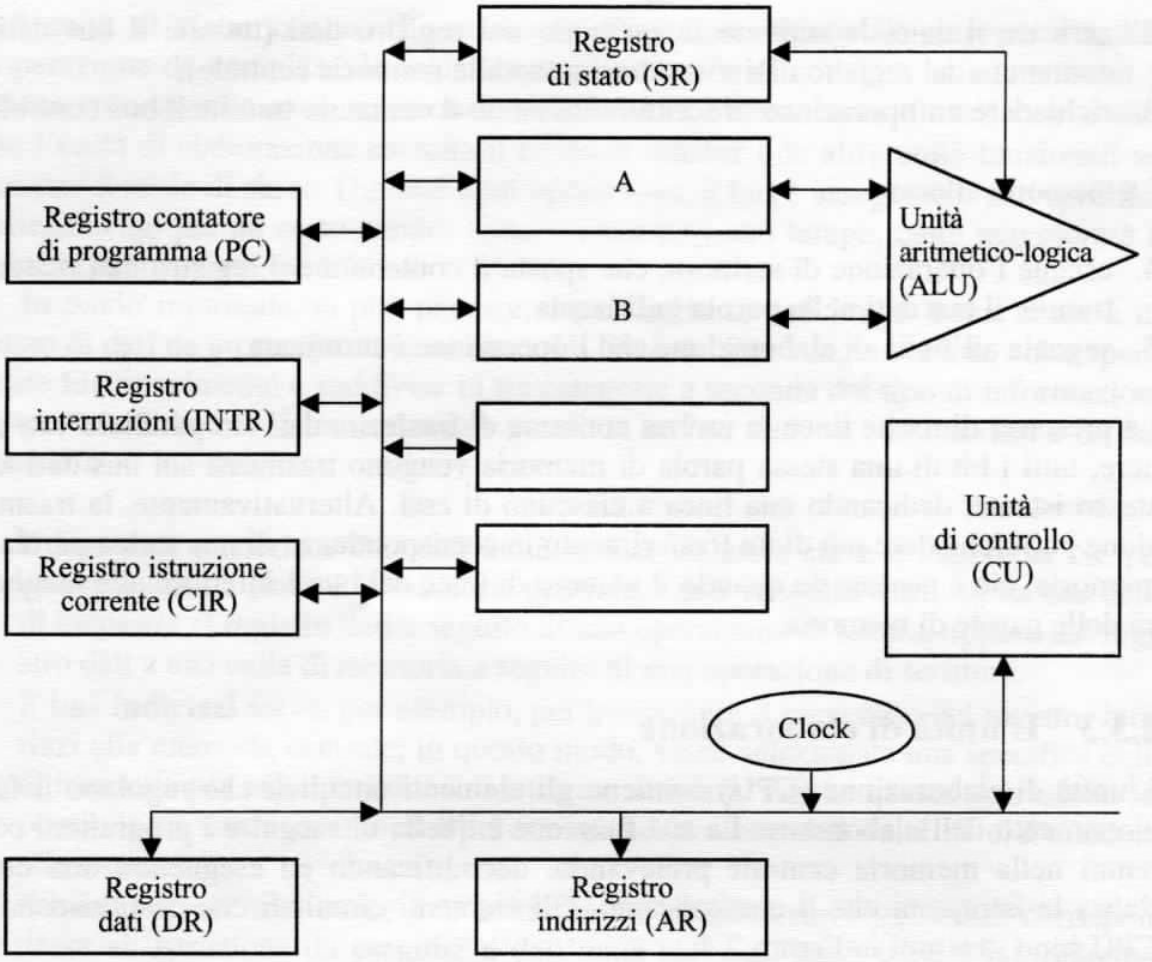

Figura 2.4 Componenti dell'unità di elaborazione

- il **contatore di programma** (PC, *Program Counter*), lungo *k* bit, che contiene l'indirizzo della prossima istruzione del programma in esecuzione;
- il **registro interruzioni** (INTR, *Interrupt Register*), che contiene alcune informazioni relative allo stato di funzionamento delle periferiche; l'uso di questo registro verrà discusso nel Capitolo 13;
- i registri che contengono gli operandi e il risultato delle elaborazioni, di tipo aritmetico o logico, svolte dall'unità aritmetico-logica (**A** e **B** in figura);
- un numero piuttosto grande di **registri di lavoro**, del tutto analoghi a celle di memoria ma che possono essere letti e scritti in modo molto più rapido senza ricorrere a operazioni di lettura e scrittura; tali registri contengono dati o istruzioni di uso frequente, oppure risultati intermedi delle elaborazioni.

L'unità aritmetico-logica può essere anche molto sofisticata e capace di operazioni complesse. In questo testo descriviamo una semplice ALU dotata di due soli operandi e capace di eseguire le classiche operazioni di tipo numerico (somma, sottrazione, prodotto, divisione). Convenzionalmente, chiameremo A e B i registri che contengono gli operandi. La ALU viene messa in azione dall'unità di controllo, che specifica un codice operativo corrispondente all'operazione da svolgere. In precedenza, i registri A e B vengono caricati con i valori dei due operandi. L'esecuzione di un'ope-

razione richiede un certo tempo, misurato dall'orologio di sistema, dopo il quale il registro A è caricato con il risultato dell'operazione; nel caso della divisione intera, il resto della divisione è caricato nel registro B, mentre per le altre operazioni il contenuto del registro B non è definito.

- Un registro particolare, detto **registro di stato** (SR, *State Register*), riporta in alcuni suoi bit indicazioni relative al risultato delle operazioni svolte dalla ALU. Tra di essi, ricordiamo:
  1. il **bit di carry**, che indica la presenza di un riporto (*carry-over*);
  2. il **bit zero**, che indica la presenza di un valore nullo nel registro A;
  3. il **bit di segno**, che riporta il segno del risultato di un'operazione aritmetica;
  4. il **bit di overflow**, che consente di rilevare la condizione di overflow che si verifica quando il risultato dell'ultima operazione aritmetica eseguita dalla ALU supera il valore 2^h, massimo valore che può essere rappresentato nel registro A, supposto che questo sia lungo h bit.

  Quando risulta una condizione di overflow, il bit di overflow indica che il registro A contiene un risultato errato rispetto all'ultima operazione aritmetica eseguita dalla ALU. I bit precedenti consentono operazioni di confronto fra due operandi. Queste operazioni possono essere materialmente eseguite sottraendo il secondo operando dal primo e controllando il risultato, come mostrato nell'Esempio 2.2 alla fine di questo capitolo.

È bene sottolineare infine, ancora una volta, come l'*unità di controllo* abbia il compito di coordinare l'intero sistema: interpretando le istruzioni del programma in esecuzione essa ne controlla l'esecuzione nella giusta sequenza. Sotto la sua direzione le informazioni vengono trasferite o elaborate tramite la ALU internamente alla CPU, lette e scritte in memoria, scambiate con il mondo esterno.

2.3.4 Interfacce di ingresso/uscita

Le interfacce di ingresso/uscita costituiscono gli elementi circuitali che consentono il collegamento dell'elaboratore con le varie periferiche. Un'interfaccia contiene registri per inviare comandi alla periferica, scambiare dati e controllare il funzionamento della periferica. Le interfacce sono molto diverse a seconda del tipo di periferica considerata (terminali, stampanti, dispositivi di memoria di massa, strumenti di misurazione, lettori ottici, plotter, sensori e attuatori di robot ecc.). Inoltre, le periferiche possono essere più o meno "intelligenti", dotate cioè di proprie unità di controllo capaci di convertire ed elaborare dati; ovviamente, interfacce intelligenti possono sottrarre compiti all'unità di elaborazione. Per esempio, una tipica evoluzione dell'architettura di von Neumann consiste nel dedicare alcune unità di elaborazione al solo controllo delle operazioni di ingresso/uscita. Nel seguito, ci limitiamo a caratterizzare una interfaccia standard elementare. Tale interfaccia contiene in genere i seguenti elementi:

- un **registro dati** della periferica (PDR, *Peripherical Data Register*), per scambiare dati con la periferica. Lo scambio può avvenire verso la periferica (per esempio, con una stampante) oppure verso l'elaboratore (per esempio, con i lettori ottici). Un terminale ha due registri dati distinti, uno che consente all'elaboratore di acquisire dati dalla tastiera, e uno che consente di produrre dati da visualizzare sul video;
- un **registro comando della periferica** (PCR, *Peripherical Command Register*) che contiene il comando che la periferica stessa dovrà eseguire;
- una **informazione sullo stato** della periferica (per esempio, per una stampante lo stato può essere: pronta a ricevere un nuovo dato, occupata a stampare, oppure in una condizione di errore per mancanza di carta o di inchiostro).

Il registro dati viene collegato al bus dati, mentre il registro comando viene collegato al bus controlli; l'informazione sullo stato può venire trasferita in un apposito **registro di stato della periferica** (PSR, *Peripherical Status Register*) e letta a "co-mando" dall'unità di elaborazione (utilizzando il bus), oppure può essere collegata con circuiti elettrici speciali all'unità di elaborazione (nel qual caso questa informazione concorre a riempire il contenuto del registro interruzioni, INTR). Vedremo nel Capitolo 13, dedicato ai sistemi operativi, come vengono gestite le operazioni di ingresso/uscita.

2.4 Esecuzione dei programmi

In questo capitolo abbiamo introdotto l'architettura di von Neumann allo scopo di presentare un modello operativo del calcolatore, ossia per illustrare come quest'ultimo possa eseguire semplici algoritmi. Ci limiteremo ora a mostrare alcuni esempi, senza definire uno specifico linguaggio macchina.

Per iniziare l'esecuzione di un programma, la macchina di von Neumann necessita che venga caricata in memoria centrale la forma binaria del programma. La **forma binaria** di un programma, mostrata in Figura 2.5, è naturalmente una sequenza di parole binarie (come tutte le informazioni contenute nella memoria di un elaboratore). La figura mostra anche l'allocazione del programma nella memoria principale, supponendo di partire dalla cella numero zero (la prima cella della memoria). Per eseguire un programma, la macchina di von Neumann acquisisce le istruzioni di programma dalla memoria centrale e le esegue ripetendo una sequenza di operazioni nella CPU. Si noti che il programma è costituito da due parti distinte: la prima contiene le istruzioni codificate, la seconda contiene i dati (in ingresso, in uscita, calcolati e temporanei). Le due parti sono divise dalla speciale istruzione "halt"; quando la macchina esegue questa istruzione l'esecuzione del programma si arresta.

Ogni istruzione viene eseguita in tre fasi: l'acquisizione dalla memoria centrale, l'interpretazione (decodifica) e l'esecuzione.

La fase di acquisizione, chiamata anche **fetch phase**, si svolge a sua volta in quattro passi; ogni passo corrisponde a un trasferimento di dati fra i registri della CPU e/o allocazioni specifiche della memoria centrale. La fase consta dei passi illustrati di seguito:

0100000000010000	leggi un valore in ingresso e ponilo nella cella numero 16 (variabile *a*)
0100000000010001	leggi un valore e ponilo nella cella numero 17 (variabile *b*)
0100000000010010	leggi un valore e ponilo nella cella numero 18 (variabile *c*)
0100000000010011	leggi un valore e ponilo nella cella numero 19 (variabile *d*)
0000000000010000	carica il registro A con il contenuto della cella 16 (valore *a*)
0001000000010001	carica il registro B con il contenuto della cella 17 (valore *b*)
0110000000000000	somma i contenuti dei registri A e B
0010000000010100	immagazzina il contenuto del registro A nella cella numero 20 (risultato parziale)
0000000000010010	carica il registro A con il contenuto della cella 18 (valore *c*)
0001000000010011	carica il registro B con il contenuto della cella 19 (valore *d*)
0110000000000000	somma i contenuti dei registri A e B
0001000000010100	immagazzina il contenuto del registro A nella cella numero 20 (risultato)
0101000000010100	scrivi in output il contenuto della cella numero 20 (risultato)
1101000000000000	arresta l'esecuzione del programma
.....	spazio per la *a* (cella 16)
.....	spazio per la *b* (cella 17)
.....	spazio per la *c* (cella 18)
.....	spazio per la *d* (cella 19)
.....	spazio per il risultato (cella 20)

(*a*)

0100000000010000	cella numero 0
0100000000010001	cella numero 1
0100000000010010	cella numero 2
0100000000010011	cella numero 3
0000000000010000	cella numero 4
0001000000010001	cella numero 5
0110000000000000	cella numero 6
0010000000010100	cella numero 7
0000000000010010	cella numero 8
0001000000010011	cella numero 9
0110000000000000	cella numero 10
0001000000010100	cella numero 11
1000000000000000	cella numero 12
0010000000010100	cella numero 13
0101000000010100	cella numero 14
1101000000000000	cella numero 15
	cella numero 16 riservata alla variabile *a*
	cella numero 17 riservata alla variabile *b*
	cella numero 18 riservata alla variabile *c*
	cella numero 19 riservata alla variabile *d*
	cella numero 20 riservata al risultato
	celle di memoria libere

(*b*)

Figura 2.5 (*a*) Forma binaria del programma dell'Esempio 2.1. (*b*) Il programma dell'Esempio 2.1 caricato nella memoria centrale

1. Il contenuto del registro contatore di programma (PC) viene trasferito nel registro indirizzi (AR). Si noti che il registro PC contiene l'indirizzo della prossima istruzione da eseguire, che generalmente è quella che segue immediatamente l'ultima istruzione eseguita.
2. Avviene un'operazione di lettura dalla memoria centrale. Il contenuto della cella di memoria che corrisponde all'indirizzo contenuto nell'AR viene trasferito nel registro dati (DR) tramite il bus di sistema.
3. Il contenuto del registro dati (DR) è trasferito al registro di istruzione corrente (CIR).
4. Il valore del registro PC viene incrementato di 1, in modo che tale registro arriva a contenere l'indirizzo dell'istruzione che segue immediatamente quella attualmente caricata nel CIR; in tal modo viene preparata l'esecuzione della fase di acquisizione successiva. Tuttavia (come vedremo nell'Esempio 2.2), è possibile che durante l'esecuzione dell'istruzione corrente venga memorizzato in PC un indirizzo differente, modificando in tal modo l'esecuzione "sequenziale" del programma. Questo genere di istruzioni viene denominato **branch**.

La Figura 2.6*a* mostra il comportamento della macchina di von Neumann durante la fase di acquisizione della prima istruzione del programma riportato nell'Esempio 2.1.

La fase successiva, detta **fase di interpretazione**, è dedicata alla decodifica dell'istruzione. Questa comporta l'analisi del contenuto del registro CIR per scoprire quale operazione deve essere eseguita. In questa fase viene analizzato soltanto il codice operativo dell'istruzione corrente. Con riferimento alla Figura 2.6*b*, il codice operativo dell'istruzione contenuta nella cella numero 0 del programma dell'Esempio 2.1 viene interpretato dalla macchina come operazione di lettura, ossia: "trasferisci i dati disponibili nel registro dati della periferica in una cella della memoria centrale".

Infine, la **fase di esecuzione**, che è differente per ogni operazione, consiste nell'esecuzione dell'operazione stessa. Le operazioni supportate da ogni calcolatore sono differenti; comprendono il trasferimento dati da o alla memoria e da o alle periferiche, e operazioni supportate dalla ALU e dai vari registri macchina. La Figura 2.6*c* illustra il comportamento della macchina di von Neumann nella fase di esecuzione della prima istruzione del programma riportato nell'Esempio 2.1; questa parte della figura risulterà più comprensibile dopo la lettura di tale esempio.

*Esempio 2.1

Vogliamo calcolare il valore dell'espressione $(a + b) * (c + d)$, leggendo i valori delle variabili da un dispositivo di ingresso e scrivendo i risultati sul dispositivo di uscita.

Un algoritmo generale che assolve tale compito è il seguente:

1. leggi i valori a, b, c e d dal dispositivo di ingresso;
2. somma i valori di a e b;
3. salva il risultato parziale in memoria;

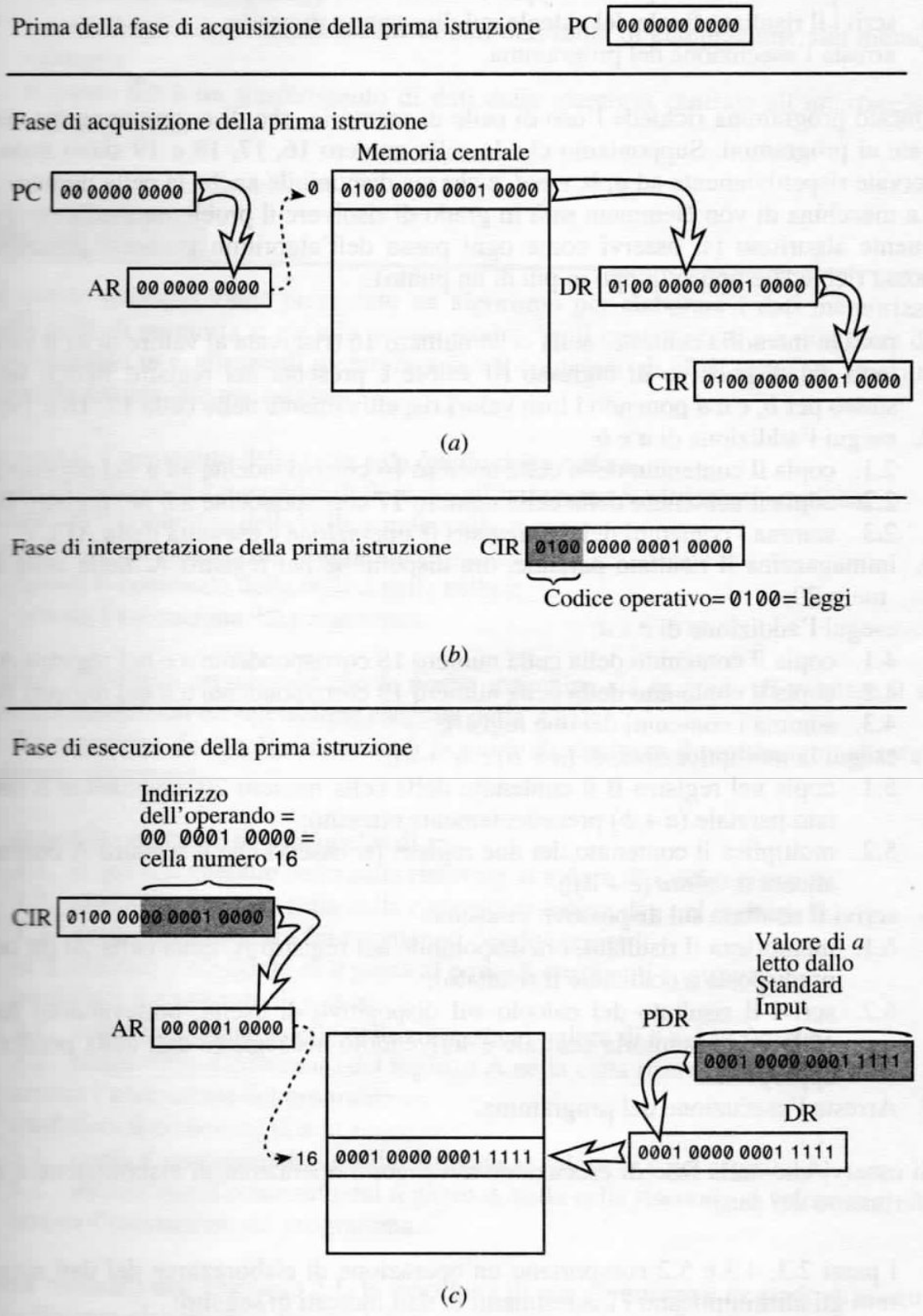

Figura 2.6 Comportamento della macchina di von Neumann durante l'elaborazione della prima istruzione del programma riportato in Esempio 2.1

4. somma i valori di *c* e *d*;
5. moltiplica il risultato parziale appena ottenuto con quello precedentemente salvato;
6. scrivi il risultato finale del calcolo sul dispositivo di uscita;
7. arresta l'esecuzione del programma.

Questo programma richiede l'uso di celle di memoria, che di norma vengono riservate ai programmi. Supponiamo che le celle numero 16, 17, 18 e 19 siano state riservate rispettivamente ad *a*, *b*, *c* e *d*, e che sia disponibile anche la cella numero 20. La macchina di von Neumann sarà in grado di risolvere il problema mediante il seguente algoritmo (si osservi come ogni passo dell'algoritmo generale precedente possa richiedere uno sviluppo in più di un punto):

1. poni in memoria centrale, nella cella numero 16 (riservata al valore di *a*) il valore letto dal dispositivo di ingresso (il valore è presente nel registro PDR); fai lo stesso per *b*, *c* e *d* ponendo i loro valori rispettivamente nelle celle 17, 18 e 19;
2. esegui l'addizione di *a* e *b*:
   2.1. copia il contenuto della cella numero 16 corrispondente ad *a* nel registro A;
   2.2. copia il contenuto della cella numero 17 corrispondente a *b* nel registro B;
   2.3. somma i contenuti dei due registri (l'operazione è eseguita dalla ALU);
3. immagazzina il risultato parziale, ora disponibile nel registro A, nella cella numero 20;
4. esegui l'addizione di *c* e *d*:
   4.1. copia il contenuto della cella numero 18 corrispondente a *c* nel registro A;
   4.2. copia il contenuto della cella numero 19 corrispondente a *d* nel registro B;
   4.3. somma i contenuti dei due registri;
5. esegui la moltiplicazione di $(a + b)$ e $(c + d)$:
   5.1. copia nel registro B il contenuto della cella numero 20, contenente il risultato parziale $(a + b)$ precedentemente ottenuto;
   5.2. moltiplica il contenuto dei due registri [si osservi che il registro A contiene ancora il valore $(c + d)$];
6. scrivi il risultato sul dispositivo di uscita:
   6.1. memorizza il risultato, ora disponibile nel registro A, nella cella 20 (la cella predisposta a contenere il risultato);
   6.2. scrivi il risultato del calcolo sul dispositivo di uscita, prelevandolo dalla cella 20 in memoria centrale e scrivendolo nel registro dati della periferica appropriata.
7. Arresta l'esecuzione del programma.

Si osservi che nella fase di esecuzione avvengono operazioni di elaborazione e trasferimento dei dati:

- i passi 2.3, 4.3 e 5.2 comportano un'operazione di elaborazione dei dati mentre tutti gli altri implicano i trasferimenti di dati indicati di seguito:
- il passo 1 è un trasferimento di dati dall'interfaccia di una periferica alla memoria centrale;

- i passi 2.1, 2.2, 4.1, 4.2 e 5.1 sono trasferimenti di dati dalla memoria centrale all'unità di elaborazione;
- i passi 3 e 6.1 sono trasferimenti di dati dall'unità di elaborazione alla memoria centrale;
- il passo 6.2 è un trasferimento di dati dalla memoria centrale all'interfaccia di una periferica.

*Esempio 2.2

In questo esempio viene presentato un algoritmo per elaborare i dati memorizzati nelle celle di memoria x, y e z in questo modo: "se il contenuto di x è maggiore di y memorizzalo in z, altrimenti memorizza in z il contenuto di y". Il seguente algoritmo generale esegue tale operazione:

1. sottrai il contenuto della cella y da quello della cella x;
2. se la differenza è maggiore di 0, passa all'istruzione 5, altrimenti continua;
3. sposta il contenuto della cella y nella cella z;
4. arresta l'esecuzione del programma;
5. sposta il contenuto della cella x nella cella z;
6. arresta l'esecuzione del programma.

Questo algoritmo presuppone che la nostra macchina sia in grado di valutare il segno del risultato di un'operazione eseguita dalla ALU.

La macchina di von Neumann sarà in grado di risolvere il problema mediante il seguente algoritmo:

1. sottrai il contenuto di y da quello di x:
   1.1. copia il contenuto della cella riservata al valore di x nel registro A;
   1.2. copia il contenuto della cella riservata al valore di y nel registro B;
   1.3. esegui la differenza fra i contenuti dei due registri;
2. se il risultato è maggiore di 0 passa al punto 5 altrimenti continua;
3. trasferisci il contenuto di y in z:
   3.1. copia il contenuto della cella riservata al valore di y nel registro A;
   3.2. memorizza il contenuto del registro A nella cella riservata al valore di z;
4. arresta l'esecuzione del programma;
5. trasferisci il contenuto di x in z:
   5.1. copia il contenuto della cella riservata al valore di x nel registro A;
   5.2. memorizza il contenuto del registro A nella cella riservata al valore di z;
6. arresta l'esecuzione del programma.

Come mostrato nell'Esempio 2.1, la macchina di von Neumann è in grado di eseguire facilmente elaborazioni (punto 1.3) o trasferimenti di dati fra gli elementi funzionali della macchina stessa (punti 1.1, 1.2, 3.1, 3.2, 5.1 e 5.2).

Poniamo ora la nostra attenzione sull'esecuzione del punto 2. Innanzitutto, si noti che la fase di esecuzione del passo 1.3 ha stabilito il bit di segno del registro di stato

in accordo con il segno del risultato ottenuto sottraendo i contenuti dei registri A e B. L'esecuzione del punto 2 controlla il valore del bit di segno nel registro di stato; se esso indica un risultato maggiore di 0, pone l'indirizzo in memoria centrale del punto 5.1 nel registro PC, altrimenti lascia in PC il valore esistente (in riferimento all'istruzione 3.1).

Il valore del registro PC dopo l'esecuzione del punto 2 determina quale istruzione viene acquisita, decodificata ed eseguita dopo il passo 2 (l'istruzione 5.1 nel primo caso, l'istruzione 3.1 altrimenti). In questo modo la macchina di von Neumann opera una scelta fra istruzioni differenti da eseguire, come richiesto in genere dagli algoritmi.

Abbiamo dunque completato la prima descrizione dell'architettura di un elaboratore; siamo ora pronti a passare alla programmazione di alto livello in C. Maggiori informazioni sull'organizzazione e l'architettura di un calcolatore verranno date nei Capitoli 11 e 12.

Esercizi

2.1 Illustrare la relazione che intercorre fra il numero di linee del bus dati e la lunghezza di una parola.

2.2 Descrivere le caratteristiche (collegamenti, tipo e lunghezza dei registri, dimensione del bus) di una macchina dotata di un video, una tastiera, una memoria centrale di 256 kilobyte con parole lunghe 16 bit e due tipi di memorie di massa (unità a dischetto e unità a disco rigido).

2.3 Spiegare il modo in cui, nell'architettura illustrata in questo capitolo, è possibile confrontare due numeri e dire quando sono uguali.

2.4 Scrivere un algoritmo per il calcolo dell'espressione $x = a + b$, dove a e b sono le variabili in ingresso e x è la variabile di uscita. L'algoritmo deve essere eseguibile dalla macchina di von Neumann descritta in questo capitolo.

2.5 Considerando il seguente algoritmo generale:

1. leggi i valori di due variabili n e m dal dispositivo di ingresso;
2. scrivi il valore della variabile n sul dispositivo di uscita, ripetendo l'operazione di scrittura il numero di volte indicato dal valore di m;

scrivere un algoritmo eseguibile dalla macchina di von Neumann.

2.6 Descrivere le fasi di acquisizione, interpretazione ed esecuzione di tutte le istruzioni dei due esempi presentati in questo capitolo, come mostrato in Figura 2.6 per la prima istruzione dell'Esempio 2.1.