Periferiche

G. Lettieri

15 Marzo 2024

Studiamo ora le periferiche essenziali di un PC. Faremo riferimento alle vecchie periferiche del PC AT dell'IBM, che per molti anni hanno fatto parte della cosidetta *Industry Standard Architecture*, uno standard di fatto seguito da tutti i produttori di PC IBM-compatibili e cloni. Queste perifieriche hanno il vantaggio di essere più semplici da programmare dei loro equivalenti moderni. Inoltre, molti PC moderni continuano a supportarle, per mantenere la compatibilità software. In particolare, le periferiche che studiamo sono supportate dalla macchina virtuale QEMU che usiamo per sviluppare tutti gli esempi. In ogni caso i concetti base sono rimasti sostanzialmente gli stessi nel tempo.

Per ogni periferica daremo una breve descrizione del suo funzionamento interno, quindi descriveremo l'interfaccia visibile al programmatore, almeno quanto basta per poter programmare dei semplici esempi. Il codice degli esempi può essere scaricato da

`https://calcolatori.iet.unipi.it/resources/esempiIO-7.4.tar.gz`.

Tutti gli esempi fanno uso della libreria `libce`, scaricabile dallo stesso sito.

1 Tastiera

Lo scopo della tastiera è di rilevare i tasti premuti e rilasciati e di comunicarli al PC. A (quasi) ogni tasto fisico sono associati due codici numerici: il *make code*, che indica che il tasto è stato premuto, e il *break code*, che indica che il tasto è stato rilasciato. Questi codici sono associati al tasto e non alla sua funzione: per esempio, i due tasti shift hanno codici diversi. È il software che assegna (liberamente) un significato ai tasti.

Per svolgere il proprio compito, le tastiere contengono tipicamente una matrice di collegamenti elettrici. Nelle tastiere più comuni (ed economiche) questa è realizzata con tre fogli di plastica sovrapposti, chiamiamoli 1, 2 e 3. Sul foglio 1 sono tracciati dei collegamenti verticali e sul foglio 3 dei collegamenti orizzontali. Le tracce si incrociano in corrispondenza di ciascun tasto, ma sono tenute separate dal foglio 2, che si trova in mezzo agli altri due. Il foglio 2, però, contiene un buco in corrispondenza di ogni tasto (e dunque di ogni incrocio). Ogni volta che si preme un tasto si mette in collegamento, attraverso il buco,

una traccia verticale del foglio 1 con una traccia orizzontale del foglio 3, semplicemente esercitando una pressione sul foglio superiore tramite un bastoncino di gomma che si trova sotto il tasto.

La matrice è monitorata da un *microcontrollore* che si trova a bordo della tastiera (originariamente un Intel 8042). Un microcontrollore è un piccolo computer, con una sua CPU, una sua RAM, una ROM e delle porte di I/O, interamente contenuto in un unico chip. Le porte di I/O del microcontrollore sono direttamente collegate a dei piedini del chip. Nella tastiera, i piedini di I/O del microcontrollore sono collegati alle tracce verticali e orizzontali della matrice di cui sopra. Il microcontrollore è programmato in modo da inviare un segnale su una colonna alla volta e leggere lo stato di tutte le righe della matrice. Se un tasto di quella colonna è premuto, il segnale ritornerà sulla corrispondente riga: in questo modo il controllore può rilevare la pressione di un tasto[1]. Il microcontrollore passa poi alla colonna successiva, e così via. La scansione viene ripetuta migliaia di volte al secondo, cosa che è sufficiente a non perdere le battute anche del più veloce dattilografo. Per rilevare quando un tasto è stato rilasciato, il microcontrollore ricorda lo stato di ogni tasto nella propria RAM. Quando un tasto cambia stato, il microcontrollore trasmette un appropriato codice di scansione al PC.

L'informazione di "tasto rilasciato" (break code) serve al software per gestire le combinazioni di tasti (se ricevo il make code di shift seguito dal make code del tasto 'a', prima di aver visto il break code del tasto shift, vuol dire che l'utente sta cercando di scrivere 'A'). Le tastiere dei PC, invece, gestiscono autonomamente i tasti *typematic*: se il controllore della tastiera si accorge che uno dei tasti typematic è sempre premuto per tutto un certo intervallo di tempo (configurabile), inizia a inviare il make code di quel tasto ripetutamente, con un certo ritmo (anch'esso configurabile)[2].

La comunicazione tra tastiera e PC può avvenire in vari modi. Nel seguito facciamo riferimento alle tastiere di tipo PS/2, la cui interfaccia di programmazione rispecchia ancora quella del PC AT. In questo caso la comunicazione tra PC e tastiera è su un cavo seriale, e all'altro capo del cavo, sulla scheda madre del PC, si trova un altro microcontrolore (originariamente anch'esso un Intel 8042), che riceve i codici, li converte nei corrispondenti make code o break code (la conversione avviene, al solito, per motivi di compatibilità software), e li rende disponibili in un registro mappato nello spazio di I/O del bus, da cui il software può poi leggerli. I make code e break code sono su 8 bit (per quasi tutti i tasti). Inoltre, il break code differisce dal corrispondente make code solo per il bit più significativo (1 per i break code e 0 per i make code).

[1] Per semplicità evitiamo di discutere i problemi che si verificano quando più di un tasto per colonna o riga sono contemporanemamente premuti (*ghosting* e *jamming*), o dei limiti al cosiddetto *roll-over*, cioè del massimo numero di tasti contemporaneamente premuti che la tastiera può correttamente rilevare e/o riportare.

[2] Non tutti i tasti sono typematic. Per esempio, i tasti shift, ctrl, alt non lo sono.

1.1 Interfaccia per il programmatore

Il programmatore interagisce esclusivamente con il microcontrollore a bordo del PC, tramite una interfaccia che espone quattro registri: un registro di lettura, RBR, un registro di stato STR, un registro di scrittura, TBR, e un registro di controllo, CMR, tutti da 8 bit. Questi registri occupano solo due indirizzi dello spazio di I/O, in modo da risparmiare piedini di indirizzo nell'interfaccia: RBR e TBR sono entrambi all'indirizzo 0x60, mentre STR e CMR sono entrambi all'indirizzo 0x64. Questo è possibile perché RBR e STR sono di sola lettura, mentre TBR e CMR sono di sola scrittura.

I bit 0 e 1 del registro STR fungono da flag "busy" per i registri RBR e TBR, rispettivamente. In particolare, il programmatore deve assicurarsi di leggere RBR solo se il bit 0 di STR è 1, per essere sicuro di star leggendo un nuovo codice.

Tramite il microcontrollore a bordo del PC è possibile anche *inviare* comandi alla tastiera, per esempio per accendere o spegnere i led, o configurare i parametri del typematic. Questo è uno degli scopi dei registri CMR e TBR. Un altro scopo, non legato alla tastiera, è quello di iniziare il reset del PC da software: uno dei piedini del microcontrollore è collegato alla circuteria di reset del PC (e in particolare al piedino di reset del processore) e per attivarlo è sufficiente scrivere 0xFE in CMR (la funzione `reboot()` di `libce` fa esattamente questo). Vedremo poi un altro utilizzo di questi registri quando parleremo delle interruzioni.

La cartella `esempiIO` contiene due programmi che utilizzano l'iterfaccia della tastiera: `tastiera-1` e `tastiera-2`.

L'esempio `tastiera-1` legge ciclicamente un nuovo codice e lo stampa sul video in binario. L'esempio fa uso di alcuni tipi e funzioni definite in `libce`:

1. `ioaddr` è un **`typedef`** per **`unsigned short`** (16 bit) ed è utilizzato per definire gli indirizzi nello spazio di I/O;

2. `inputb()` serve a leggere un byte dallo spazio di I/O, all'indirizzo passato come argomento;

3. `char_write()` serve a mostrare un carattere sullo schermo.

La funzione `inputb()` è definita in assembly (nel file `64/inputb.s` nei sorgenti di `libce`) in modo da poter utilizzare l'istruzione **`inb`**, che è sconosciuta al compilatore C++. La funzione `char_write()` la vedremo quando parleremo del video. La funzione `get_code()` legge ciclicamente STR fino a quando non trova il bit 0 settato a 1, quindi legge il nuovo codice da RBR. Si noti che una funzione sostanzialmente identica è definita anche in `libce`, e negli esempi successivi useremo direttamente quella offerta dalla libreria.

Il programma `tastiera-2` vuole mostrare un semplice esempio di operazioni tipicamente svolte dal software, come convertire i codici letti dalla tastiera in codici ASCII, tenendo conto dello stato di tasti modificatori come lo shift, e farne l'eco sul video. In particolare, la funzione `char_read()` legge un nuovo codice dalla tastiera e tiene traccia dello stato dello shift sinistro nella variabile

globale `shift`. Il **`while`** serve ad ignorare i codici dello shift, che abbiamo già gestito, e i break code degli altri tasti, che non ci interessano (tutti i break code sono maggiori di 0x80). La funzione `conv()` provvede poi ad associare un codice ASCII al make code ricevuto, usando la tabella `tabmin` o `tabmai`, in base allo stato corrente del tasto shift. La libreria `libce` contiene versioni di `conv()` e `char_read()` identiche a quelle mostrate in questo esempio; contiene anche le tabelle `tab`, `tabmin` e `tabmai`, ma con qualche codice in più.

2 Video

L'interfaccia video ruota attorno all'idea della *memoria video*. Si tratta di una memoria destinata a contenere una descrizione di ciò che deve apparire sullo schermo. Alla memoria video accedono concorrentemente sia il software (normalmente in scrittura), sia un *controllore video*, che la legge completamente (per esempio 60 o 90 volte al secondo) e ne interpreta il contenuto per generare i segnali per il monitor (analogici nei vecchi monitor VGA, digitali nei monitor moderni). Tranne che nelle schede video più economiche, la memoria video si trova a bordo della stessa scheda video che contiene anche il controllore ed è mappata nello spazio di indirizzamento di memoria del bus. Questo vuol dire che, per il programmatore, accedere alla memoria video è sostanzialmente come accedere ad una struttura dati. A livello assembly è possibile usare tutte le istruzioni che ammettono un operando in memoria (tipicamente `mov`).

Oltre a scrivere e leggere dalla memoria video, il software può dialogare anche con il controllore stesso. Quest'ultimo ha una interfaccia con vari registri che, nella scheda che vedremo, sono accessibili nello spazio di I/O. Scrivendo opportunamente in questi registri è possibile, in particolare, selezionare la *modalità* video, che stabilisce come deve essere interpretato il contenuto della memoria video. Esistono due modalità principali, con varie sotto-modalità:

- *Modalità testo*: in questo caso la memoria video contiene i codici (per es. ASCII) dei caratteri che devono essere visualizzati sullo schermo, più eventualmente altri attributi come il colore dei singoli caratteri;
- *Modalità grafica*: in questo caso la memoria video serve a specificare il colore di ciascun pixel dello schermo.

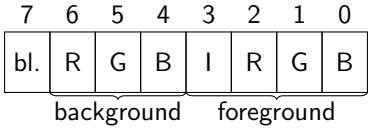
Nel caso della modalità testo, il controllore video ha anche bisogno di un *font* che gli dica come disegnare sullo schermo i vari caratteri. Normalmente uno o più font sono contenuti in una ROM a bordo della scheda video. Le varie sottomodalità della modalità testo si distinguono per il numero delle righe e colonne in cui è suddiviso lo schermo. Le varie sotto-modalità della modalità grafica si distinguono invece per la risoluzione (normalmente espressa in pixel orizzontali per pixel verticali) e per il numero di colori possibili per ciascun pixel. Lo standard SVGA (Super Video Graphics Array) definisce i dettagli di varie modalità, sia testo che grafiche, che le schede che supportano lo standard

devono offrire. Nel seguito esaminiamo la modalità testo 80×25 (80 righe per 25 colonne), che è quella che tradizionalmente le schede video per PC offrono di default, e le modalità grafiche di tipo *framebuffer*, che sono le più semplici da usare.

Il controllore contiene un gran numero di registri interni, ma, per risparmiare piedini, espone al programmatore due soli registri: IND e DAT. Tramite questi registri il programmatore può accedere *indirettamente* a qualunque registro interno. Ogni registro interno è identificato da un indice e, per accedervi, il programmatore deve prima scrivere l'indice desiderato nel registro IND; a quel punto il registro DAT diventa una "finestra" sul registro interno selezionato: scrivendo e leggendo da DAT si legge e scrive nel registro interno.

2.1 Modalità testo

La scheda video emulata da QEMU si trova all'avvio in modalità testo 80×25. In questa modalità lo schermo è idealmente diviso in una matrice di 80 per 25 posizioni, ciascuna delle quali può visualizzare un carattere. La memoria video è organizzata in una corrispondente matrice di 80×25 elementi, memorizzata per righe. Il primo elemento della matrice si trova all'indirizzo 0xb8000 e corrispondente alla posizione in alto a sinistra dello schermo. Ciascun elemento della matrice è grande due byte: il byte meno significativo contiene il codice del carattere da visualizzare, mentre il byte più significativo contiene l'*attributo colore*. Per quanto riguarda il codice del carattere, normalmente le schede video dei PC accettano un qualunque codice ASCII (7 bit) eventualmente esteso a 8 bit per includere altri caratteri come, in Italiano, le lettere accentate. L'attributo colore, invece, ha il seguente formato:



dove i bit contrassegnati con *foreground* selezionano il colore di primo piano su 4 bit (codificato come Intensity, Red, Green, Blue) e i bit contrassegnati con *background* selezionano il colore di sfondo (codificato come Red, Green, Blue). Il bit 7 (*blinking*), se attivo, rende il carattere lampeggiante, ma questa funzionalità non è emulata da QEMU. Quindi, per esempio, il byte 01001011 (4B in esadecimale) seleziona un colore azzurro chiaro (I+G+B) su sfondo rosso (R).

La cartella `esempiIO` contiene due esempi sul video in modalità testo: `video-testo-1` e `video-testo-2`.

L'esempio `video-testo-1` si limita a inizializzare tutta la memoria video con lo stesso carattere e lo stesso attributo colore. Il tipo `natw`, utilizzato nel codice, è definito in `libce` come un **`typedef`** per **`unsigned short`**, ed è dunque grande due byte, quanto ciascun elemento della memoria video. Per accedere alla memoria video direttamemte dal C++ è sufficiente dichiarare un

puntatore a `natw` (chiamato `video` nel codice) e inizializzarlo con il valore numerico, noto, del primo indirizzo della memoria video (0xb8000, come detto). Per farlo accettare dal compilatore è però necessario un cast, visto che stiamo cercando di assegnare un intero ad un puntatore. Fatto questo, `video` può essere usato come un normale array di `natw`. Gli elementi da 0 a 79 corrispondono alla prima riga di caratteri, quelli da 80 a 159 alla seconda, e così via. Ogni `natw` deve contenere l'attributo colore nel byte più significativo e, nel byte meno significativo, il codice ASCII del carattere da visualizzare. Dopo aver inizializzato la memoria video, il programma attende che l'utente prema ESC, chiamando ripetutamente la funzione `char_read()` definita in `libce`, analoga a quella vista in `tastiera-2`.

L'esempio `video-testo-2` mostra come realizzare la funzione usata negli esempi precedenti per mostrare un carattere sul video, `char_write()`. La funzione tiene traccia, nelle variabili `x` e `y`, della posizione in cui dovrà apparire il prossimo carattere, in modo da emulare un "terminale a stampante", che stampa i caratteri uno a fianco all'altro. La funzione si preoccupa anche di andare automaticamente alla riga successiva quando si raggiunge l'ultima colonna, e di mostrare lo "scroll" verso l'altro del contenuto del monitor quando si raggiunge l'ultima riga. Lo scroll è realizzato copiando ogni riga dello memoria video nella sua riga superiore, quindi riempendo di spazi la riga in fondo. La funzione gestisce anche i caratteri di a-capo, che si limitano a spostare la posizione del prossimo carattere, senza stampare niente. Infine, la funzione sfrutta la possibilità, offerta dal controllore video, di mostrare un *cursore* nella posizione del prossimo carattere. Per farlo è sufficiente scrivere la posizione desiderata in due registri interni del controllore, cosa che è svolta dalla funzione `cursore()`.

2.2 Modalità grafica

Per attivare la modalità grafica, visto che la scheda che stiamo considerando parte in modalità testo, è necessario impostare un gran numero di registri interni. Questa operazione è molto complessa, tanto che molte schede contengono una ROM in cui sono presenti delle funzioni che svolgono questo compito. Noi "bareremo" un po', in quanto la scheda emulata da QEMU, a sua volta presa dall'emulatore Bochs, può essere configurata molto più facilmente, sostanzialmente scrivendo la risoluzione desiderata in alcuni registri. Queste operazioni sono svolte dalla funzione `bochsvga_config()`, definita in `libce`. La funzione riceve il numero di colonne e righe di pixel desiderato, mentre il numero di colori è fissato a 256. La funzione restituisce l'indirizzo del *framebuffer*, che è l'analogo della memoria video in modalità testo, solo che questa volta l'array contiene una posizione per ciascun pixel dello schermo (nella risoluzione selezionata). Ciascun elemento dell'array è grande un byte, in modo da poter contenere uno qualunque dei 256 colori possibili. Il tipo del valore restituito dalla funzione è direttamente un puntatore a `natb` (**`unsigned char`**), in modo che lo si possa usare senza dover fare un cast.

A questo punto, per disegnare qualcosa sullo schermo, è sufficiente colorare opportunamente i pixel desiderati, semplicemente scrivendo il codice del colore

nella posizione corrispondente del framebuffer. Si vedano i programmi `svga-1`, `svga-2` e `sgva-3` per tre semplici esempi: uno che colora tutto lo schermo di rosso, uno che disegna un bordo e degli assi, e uno che aggiunge una parabola e una retta.

3 Timer (conteggio)

Il PC AT conteneva una interfaccia di *conteggio* usata per vari scopi. Molti PC moderni continuano a emulare questa interfaccia, e la troviamo anche in QEMU. Queste interfacce decrementano un contatore interno ogni volta che si verifca un evento, segnalato da un inpulso su un loro piedino di ingresso. In particolare, se questo piedino è collegato ad un generatore di clock, l'interfaccia permette di misurare il passaggio del tempo. Il valore iniziale del contatore può essere impostato via software, scrivendo in un apposito registro dell'interfaccia. La scrittura, normalmente, funge anche da *trigger* che avvia il conteggio. Quando il contatore arriva a zero, l'interfaccia genera un segnale su un piedino di uscita, con varie possibili forme d'onda. Queste interfacce possono funzionare a *ciclo singolo* o a *ciclo continuo*. Nel secondo caso l'interfaccia, a fine conteggio, ricarica automaticamente il contatore e fa ripartire il conteggio.

Il chip che si trovava a bordo del PC AT era l'Intel 8254, che conteneva tre interfacce di conteggio indipendenti, chiamate 0, 1 e 2, ciascuna con un contatore a 16 bit. Tutte e tre le interfacce erano collegate a un generatore di clock a 1,190 MHz. Ciascuna interfaccia era indipendentemente configurabile per selezionare, per esempio, il ciclo singolo o continuo, oppure il tipo di forma d'onda da generare (impulso, onda quadra, ...). L'interfaccia 0 era collegata, come vedremo, al controllore delle interruzioni. L'interfaccia 1 era utilizzata per il refresh della memoria dinamica (e non è più emulata nei PC moderni). Infine, l'interfaccia 2 era collegata all'unico dispositivo audio fornito, per default, dal PC AT: un "cicalino" in grado di produrre dei *beep*. Si tratta in pratica di un piccolo altoparlante, una membrana che può essere messa in vibrazione agendo su un elettromagnete.

3.1 Interfaccia per il programmatore

Il chip 8254 contiene 13 registri da 8 bit, 4 per ogni interfaccia di conteggio e uno a comune. Per ogni interfaccia, due registri sono di sola scrittura e permettono di impostare la parte alta e bassa del contatore (CTR_LSB e CTR_MSB); due registri sono di sola lettura e permettono di leggere lo stato corrente del contatore (STR_LSB e STR_MSB). Il registro comune (CWR) serve principalmente a configurare le interfacce. Il chip, però, possiede solo 2 piedini di indirizzo, e dunque occupa solo 4 indirizzi dello spazio di I/O, in particolare gli indirizzi 0x40–0x43. L'indirizzo 0x43 permette di accedere al registro comune, CWR. L'indirizzo 0x40 è utilizzato per tutti e 4 i registri dell'interfaccia 0, l'indirizzo 0x41 per i 4 dell'interfaccia 1 (non più utilizzato, come abbiamo detto), e l'indirizzo 0x42 per i 4 dell'interfaccia 2. Scrivendo all'indirizzo 0x40 si acce-

de alternativamente al registro CTR_LSB e CTR_MSB del contatore 0, mentre leggendo si accede alternativamente ai registri STR_LSB e STR_MSB, e così per gli altri due contatori. In pratica, per impostare il valore completo di un contatore, è necessario eseguire due scritture consecutive allo stesso indirizzo di I/O.

Dal momento che non conosciamo ancora le interruzioni, vediamo per il momento un esempio che usa il cicalino, e dunque il contatore 2. L'idea è che questo contatore deve essere programmato in modo da generare un'onda quadra a ciclo continuo, con un certo periodo. Questa onda quadra pilota (tramite un amplificatore e un filtro passa basso) direttamente l'elettromagnete dell'altoparlante, e dunque fa vibrare la membrana con la stessa frequenza, producendo una nota udibile. Il PC contiene anche un registro SPR, all'indirizzo di I/O 0x61, che serve a controllare più finemente l'ingresso dell'altoparlante. In particolare, il bit 1 del registro SPR è in AND con l'uscita del contatore 2, e permette dunque di silenziare l'altoparlante se posto a 0; il bit 0 di SPR, invece, è in ingresso al contatore 2 e, se posto a 0, mette in pausa il conteggio.

La cartella `esempiIO` contiene il programma `timer`, che usa questo contatore per generare un sibilo a 1000 Hz. La funzione `avvia_timer()` configura e inizializza il contatore 2. La scrittura del valore 0xB6 in CWR imposta il contatore 2 per generare un'onda quadra a ciclo continuo. Le due successive scritture (si noti che `iCTR2_LSB` e `iCTR2_MSB` sono in realtà lo stesso indirizzo) impostano il valore del contatore a 1190, in modo che il contatore arrivi a zero 1000 volte al secondo. La funzione `main` provvede anche a scrivere 3 (11 in binario) nel registro SPR, in modo da *non* mettere in pausa il conteggio e permettere al segnale del contatore di raggiungere l'altoparlante. La funzione `pause()`, definita in `libce`, scrive un messaggio sul video e aspetta che l'utente prema ESC. Al ritorno dalla funzione, la funzione `main` scrive 0 in SPR, in modo da silenziare l'altoparlante, e quindi termina.

4 Hard disk

L'hard disk è un esempio di dispositivo *a blocchi*. Con questo termine ci si riferisce a dispositivi di memorizzazione in cui l'informazione è organizzata in unità relativamente grandi, dette appunto blocchi, che sono anche l'unità di trasferimento. Per esempio, negli hard disk comuni i blocchi sono tradizionalmente grandi 512 byte, e non è possibile leggere o scrivere meno di 512 byte per volta. Pur essendo dispositivi di memoria, dunque, non possono essere interfacciati direttamente con la CPU, che invece accede ad unità molto più piccole, e devono essere considerati dei dispositivi di ingresso/uscita: per poter accedere a informazioni contenute in un hard disk, il software deve tipicamente prima copiare i relativi blocchi in memoria centrale, ordinando il trasferimento tramite una interfaccia di I/O.

Internamente, gli hard disk si possono scomporre in un componente detto *drive*, che comprende i dischi, le testine di lettura/scrittura e i loro servomeccanismi, e un secondo componente detto *controllore*, che contiene la logica in

grado di pilotare il drive in base agli ordini impartiti dal software. Controllore e drive sono tipicamente venduti insieme, in un unico dispositivo, che poi deve essere collegato al resto del computer con un cavo che, nei PC moderni, è di tipo seriale.

L'informazione è memorizzata magneticamente sulle facce dei dischi, ciascuna delle quali è suddisa in tracce concentriche. Ciascuna traccia è poi suddivisa in settori. Il numero di settori per traccia aumenta man mano che ci si sposta dal centro verso la periferia del disco, in modo che la dimensione fisica dei settori resti all'incirca costante. I dischi contenuti del drive (tipicamente 2 o poco più) sono imperniati ad uno stesso asse centrale e sono tenuti costantemente in rotazione (salvo essere messi a riposo in caso di inattività prolungata, per risparmiare energia), con velocità angolari che possono andare da qualche migliaio a qualche decina di migliaia di rpm (rotazioni per minuto), a seconda del modello. Le testine di lettura/scrittura, una per ciascuna faccia di ogni disco, sono anch'esse solidali ad uno stesso asse di rotazione, in modo che in ogni istante si trovino tutte in corrispondenza della stessa traccia su ogni faccia. Le tracce che si trovano alla stessa distanza dal centro su ogni faccia, e che dunque possono essere lette/scritte senza spostare le testine, formano un cosiddetto *cilindro*. Ogni settore è dunque identificato geometricamente da tre coordinate: il numero della faccia (o, equivalentemente, della testina), il numero della traccia sulla faccia (o, equivalentemente, il numero del cilindro), il numero del settore all'interno della traccia. Per poter riferire un settore, il drive ha bisogno di conoscere queste tre coordinate. A quel punto sposterà (se necessario) le testine per portarle sul cilindro richiesto (spendendo un tempo detto di *seek*), attiverà la testina corrispondente alla faccia richiesta e aspetterà che il settore richiesto, per effetto della rotazione costante dei dischi, vi passi di sotto (spendendo un ulteriore tempo detto *latency*). Solo a questo punto il settore può essere letto o scritto. In ogni caso il trasferimento dei dati avverà da/verso un buffer interno, da cui poi dovrà essere trasferito in memoria centrale in qualche altro modo (come vedremo negli esempi più avanti).

Si noti che i tempi di seek e latency, essendo legati a fattori meccanici, sono migliorati nel tempo, ma molto più lentamente rispetto alla velocità delle componenti elettroniche, soprattutto del processore. In particolare, seek e latency sono entrambi dell'ordine di qualche *milli*secondo, contro le frazioni di *nano*secondo del ciclo di un tipico processore. Questo comporta che, in un sistema moderno, è bene cercare di leggere/scrivere da settori contigui, in modo da minimizzare lo spostamento delle testine e ridurre i tempi di latenza. In pratica, oggi, conviene usarli come dispositivi sequenziali, anche se, quando furono introdotti dall'IBM negli anni '50, avevano proprio lo scopo di permettere l'accesso casuale e superare i limiti delle unità a nastro magnetico. Oggi, soprattutto nell'elettronica di consumo, sono stati praticamente sostituiti dagli hard disk a stato solido (SSD), che non hanno parti in movimento e hanno tempi di accesso nell'ordine dei microsecondi. Sono ancora comunque molto usati nei *data centre*, perché il loro prezzo per bit è ancora ordini di grandezza inferiore a quello degli SSD, e probabilmente sarà ancora così per molti anni.

4.1 Interfaccia per il programmatore

Il software interagisce con l'hard disk solo attraverso una interfaccia che, a sua volta, interagisce con il controllore a bordo dell'hard disk. Tramite i registri dell'interfaccia il software impartisce i comandi di lettura o scrittura, specificando quale settore deve essere coinvolto, e accede al buffer interno, sia per estrarne il contenuto dopo la fine di una operazione di lettura, sia per riempirlo con i dati da memorizzare prima dell'inizio di una operazione di scrittura.

Facciamo qui riferimenmto all'interfaccia ATA (AT Attachment), che standardizza l'interfaccia che si trovava nel PC AT. Questa interfaccia prevede diversi registri a 8 bit e uno a 16 bit, tutti mappati nello spazio di I/O. Ci interessano per il momento solanto i seguenti:

- SNR (Sector Number), CNL (Cylinder Number Low), CNH (Cylinder Number High), HND (Head aNd Drive): tutti da 8 bit, permettono di specificare il (primo) settore coinvolto nell'operazione;
- SCR (Sector Counter, 8 bit): permette di specificare quanti settori (in sequenza a partire dal primo) sono coinvolti nell'operazione;
- CMD (CoMmanD, 8 bit): permette di specificare il tipo di operazione (per es., lettura o scrittura);
- BR (Buffer Register, 16 bit): permette di accedere al buffer interno, due byte alla volta;
- STS (STatuS, 8 bit): contiene due flag che permettono di sapere se la precedente operazione si è conclusa (e dunque è possibile accedere al registro BR);

Anche se l'interfaccia prevede che il programmatore identifichi un settore dandone le coordinate geometriche (testina, cilindro e settore), questo ha ormai soltanto un interesse storico, in quanto l'organizzazione interna dei dischi è oggi molto più complessa di un tempo. Quello che faremo è di usare il modo di indirizzamento detto LBA (Logical Block Address), in cui ogni settore è identificato da un numero sequenziale, a partire da 0 fino al numero di settori contenuti nell'hard disk. Questo numero va scomposto in quattro parti e scritto nei registri SNR, CNL, CNH e nei 4 bit meno significativi di HND, perché i 4 bit più significativi di questo registro hanno altre funzioni (tra cui, abilitare LBA). Si noti che in questo modo si dispone di 28 bit per identificare un settore; con settori di 512 byte questo limita la dimensione massima di un hard disk a $2^{28} \times 2^9 = 2^{37}$ byte, cioé 128 GiB, che è poco per un hard disk moderno. Per gli hard disk più grandi, lo standard prevede che il numero di settore sia spezzato in due parti di 28 e 20 bit (modalità LBA48) e scritto in due passaggi negli stessi registri, con un meccanismo simile a quello visto per il timer. L'interfaccia permette di ordinare il trasferimento di piú settori consecutivi, specificandone il numero nel registro SCR. In quel caso il controllore provvederà ad incrementare automaticamente l'LBA dopo ogni trasferimento.

La macchina QEMU che usiamo per gli esempi fornisce un hard disk ATA, emulato dal file `CE/share/hd.img` nella nostra home directory. La cartella `esempiIO` contiene due programmi che mostrano come si può ordinare una scrittura (`hard-disk-1`) o una lettura (`hard-disk-2`) da questo hard disk. Entrambi i programmi sono strutturati in modo simile. La funzione `hd_set_lba()` accetta un LBA come parametro, lo spezza in quattro parti e lo scrive nei registri SNR, CNL, CNH e HND, abilitando anche il modo LBA. Il tipo `natl` è definito in `libce` come un **`typedef`** per **`unsigned int`** (32 bit). Per semplicità la funzione non abilita la doppia scrittura, quindi `lba` deve essere minore di 2^{28}. La funzione `hd_start_cmd(lba, quanti, cmd)` avvia un trasferimento di `quanti` settori a partire da quello di indirizzo `lba`. Il parametro `cmd` deve essere uno di quelli compresi dall'interfaccia. La libreria definisce le costanti `WRITE_SEC` e `READ_SECT`, che possono essere passate alla funzione per ordinare operazioni rispettivamente di scrittura e lettura. La funzione si limita a trasferire i parametri nei corrispondenti registri (usando `hd_set_lba()` per impostare l'LBA). Una volta avviata l'operazione dobbiamo aspettare che il controllore ci dica che è possibile accedere al buffer interno (funzione `hd_wait_for_br()`) e quindi eseguire 256 scritture (o letture) in BR. Per fare quest'ultima operazione possiamo comodamente usare le istruzioni **`outs`** e **`ins`** con prefisso **`rep`**, che trasferiscono da (verso) la memoria all'indirizzo contenuto in **`rsi`** il numero di byte/word/long (a seconda del suffisso) specificato in **`rcx`**. La libreria `libce` contiene le funzioni `outputb*` e `inputb*` (dove l'asterisco sta per `b`, `w` o `l`) che utilizzano queste istruzioni. Si noti che l'operazione di controllo di STS e successiva scrittura/lettura in BR va ripetuta per ogni settore coinvolto nell'operazione.

La funzione `main` usa queste funzioni per scrivere o leggere un paio di settori a partire da un certo LBA, trasferendoli da/verso l'array `buff`. Il programma `hard-disk-2` mostra poi sul video il contenuto dell'array `buff` a lettura ultimata. Il programma `hard-disk-1` mostra solo il messaggio `OK` per dire che l'operazione si è conclusa. Possiamo verificare che la scrittura è stata effettivamente eseguita correttamente esaminando il contenuto del file che emula l'hard disk "fuori" dalla macchina virtuale, per esempio con il comando

```
hexdump -C ~/CE/share/hd.img
```

che dovrebbe mostrare una serie di caratteri 'f' a partire dall'offset 512 (200 in esadecimale), corrispondente all'inizio del settore con LBA pari a 1.