Calcolatori Elettronici: introduzione

G. Lettieri

1 Marzo 2022

1 Introduzione e richiami

La Figura 1 mostra l'architettura generale di un calcolatore moderno. Nel corso approfondiremo ogni componente. I principali argomenti di cui parleremo sono:

- protezione;
- interruzioni;
- memoria virtuale.

Questi tre argomenti ci permetteranno di parlare della *multiprogrammazione*: come eseguire "contemporaneamente" più programmi su un sistema che ha un solo processore. Studieremo tutti i dettagli che ci permetteranno di realizzare un (semplificato, ma funzionante) *nucleo di sistema operativo multiprogrammato*. Raffineremo inoltre l'architettura dell'elaboratore parlando di cache, DMA (Direct Memory Access), bus PCI. Infine, esamineremo la struttura interna di un processore moderno in grado di eseguire le istruzioni in parallelo, fuori ordine e speculativamente.

Se escludiamo questi argomenti avanzati, l'architettura di base di un calcolatore moderno resta sorprendentemente simile a quella del Manchester Baby che abbiamo visto nella lezione introduttiva. Esaminiamo di seguito i componenti principali (che comunque dovrebbero essere tutti già noti da corsi precedenti).

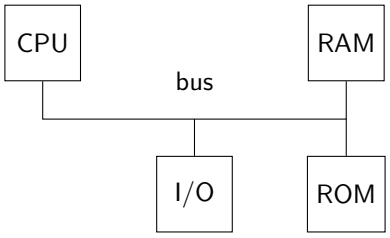

Figura 1: Architettura generale di un calcolatore moderno.

1.1 Il bus (senza DMA)

Il bus connette tra loro i vari dispositivi in modo che ciascuno possa comunicare con ciascun altro. La comunicazione deve avvenire sempre tramite operazioni di lettura o scrittura. Entrambi i tipi di operazione richiedono che venga specificato un "indirizzo". Gli indirizzi sono normalmente dei numeri in sequenza. Il nome indirizzo fu scelto per ricordare gli indirizzi delle case. La metafora vuole che immaginiamo una lunghissima via in cui ci sono case in cui abitano numeri, ciascuna con il proprio indirizzo. L'operazione di lettura all'indirizzo x chiede chi abita all'indirizzo x. L'operazione di scrittura del numero y all'indirizzo x fa si che ora y abiti all'indirizzo x (scacciando il precedente abitante). Si noti che sia gli indirizzi che gli abitanti sono numeri: è importante non confondere le due cose.

Ogni operazione è richiesta da un componente ed eseguita da un altro. Nel Manchester Baby il bus connette solo la CPU e la memoria, mentre l'I/O è sostanzialmente diverso da quello che si trova in un sistema moderno, come vedremo più avanti. La Fig. 1 mostra invece una architettura più regolare, basta su un bus comune che permette a qualunque componente di richiedere una operazione a qualunque altro. Questo diventerà importante quando parleremo di DMA, ma per il momento le uniche combinazioni possibili sono le seguenti:

1. la CPU legge o scrive sulla memoria;
2. la CPU legge o scrive sull'I/O.

Le operazioni devono essere eseguite una per volta. In ogni operazione è l'indirizzo che permette di capire se è coinvolta la memoria o l'I/O. Il discrimine può essere operato o riservando indirizzi diversi alla memoria e all'I/O, oppure definendo *spazi di indirizzamento* separati (vie diverse, nella metafora degli indirizzi delle case). Nel secondo caso, ogni operazione deve specificare anche lo spazio di indirizzamento coinvolto (memoria o I/O). Ogni operazione è vista da tutti i dispositivi collegati al bus e ciascuno di essi deve autonomamente capire se l'operazione è rivolta a lui oppure no. Per farlo deve confrontare l'indirizzo dell'operazione con gli indirizzi a lui riservati. Se l'operazione non è rivolta a lui, deve ignorarla. Se nessun dispositivo riconosce l'indirizzo possono succedere cose diverse in dipendenza dal tipo di bus: noi faremo l'ipotesi che le operazioni vengano comunque completate, con le scritture che non hanno effetto e le letture che restituiscono un valore casuale.

1.2 La memoria

L'idea è esattamente la stessa che per il Manchester Baby. Si tratta di un sistema organizzato in "celle", ciascuna composta di un certo numero di bit che è uguale per tutte le celle. Una volta che la memoria è montata sul bus, ogni cella ha un indirizzo unico che la identifica. La differenza più grande rispetto al Baby (oltre al numero di celle) è che la memoria di quest'ultimo era organizzata in celle

relativamente grandi (dette in seguito "parole"), in particolare di dimensione uguale a quella del registri. Come vedremo, le memorie moderne continuano ad essere accessibili in questo modo, ma permettono anche di accedere a unità più piccole. In particolare, nelle memorie moderne ogni singolo *byte* ha il proprio indirizzo distinto.

Per il resto, i seguenti punti continuano a essere veri:

- ogni cella contiene più precisamente una sequenza di bit; questa può essere sempre interpretata (da chi usa la memoria) come un numero naturale espresso in base due, ma il significato del contenuto di una cella dipende esclusivamente dall'uso che se ne fa;
- la memoria sa eseguire soltanto due tipi di operazioni: lettura e scrittura;
- sa eseguire una sola operazione per volta;
- per eseguire una lettura si deve specificare l'indirizzo della cella che si vuole leggere (la memoria risponde con il contenuto della cella);
- per eseguire una scrittura si deve specificare l'indirizzo della cella che si vuole scrivere, e il nuovo contenuto della cella (la memoria sostituisce il vecchio contenuto con il nuovo);
- la memoria è completamente passiva; non cambia il suo stato se non quando qualcuno ordina una operazione di scrittura;
- *tutte* le celle della memoria contengono *sempre* qualcosa, anche prima di eseguire qualunque scrittura: questo perché ogni bit è memorizzato da un dispositivo che può assumere solo uno tra due stati (0 o 1); all'accensione ci saranno zeri e uno a caso;
- ciascuna operazione richiede un tempo all'incirca costante, indipendentemente da quali altre operazioni sono state richieste precedentemente (memoria ad accesso casuale).
- nella memoria non c'è scritto cosa significhino i vari numeri, se rappresentino istruzioni, caratteri, o qualunque altra cosa.

1.3 I/O (senza interruzioni e DMA)

Anche se i dispositivi di ingresso/uscita sono tanti e vari, tutte le possibili interazioni sono trasformane in operazioni di lettura o scrittura. Queste operazioni coinvolgono particolari celle, dette registri o porte di I/O. Ogni dispositivo (tastiera, stampante, ...) sarà collegato al sistema tramite una *interfaccia*, all'interno della quale si trovano i registri. Indipendentemente dall'interfaccia a cui appartiene, nel momento in cui l'interfaccia è montata sul bus ogni suo registro acquisisce un indirizzo che lo identifica univocamente, esattamente come le celle della memoria. Le operazioni di lettura e scrittura sui registri sono del tutto simili alle analoghe operazioni sulla memoria. La differenza è che ciascuna

operazione può avere effetti collaterali, come causare la stampa di un carattere sulla carta di una stampante. Anche la lettura I/O di un registro può avere effetti collaterali, come cambiare lo stato interno di una periferica. Per esempio, leggere il codice dell'ultimo tasto premuto dal registro di ingresso della tastiera fa sì che la tastiera smetta di memorizzare quel codice; la prossima volta che il registro verrà letto, la tastiera restituirà il codice del *nuovo* ultimo tasto premuto (se ve ne sono). Ciò è completamente diverso dalla memoria, in cui il contenuto delle celle cambia solo se vi si scrive e due operazioni di lettura consecutive restituiranno sempre lo stesso valore. I dispositivi di I/O, inoltre, possono cambiare il loro stato interno in seguito alla loro interazione con il mondo esterno al calcolatore (per esempio, la tastiera memorizza il codice di un nuovo tasto se qualcuno lo preme).

Dall'interno del calcolatore, il mutamento di stato di una periferica è visibile esclusivamente dal fatto che cambia il contenuto dei registri della sua interfaccia. Le interfacce, inoltre, (in assenza dei meccanismi di interruzione e DMA) aspettano passivamente che qualcuno legga il nuovo valore dei registri. Questa è anche la principale differenza con il Manchester Baby, che abbiamo visto nell'esempio introduttivo. Nel Manchester Baby l'I/O è *direttamente* collegato alla RAM[1] Ricordiamo che l'I/O del Baby consiste nello schermo, che ci permette di vedere direttamente il contenuto della RAM, e della pulsantiera che ci permette di scrivere direttamente in RAM. Con "direttamente" intendiamo che non è necessaria (e, in questo caso, neanche possibile) la mediazione del software: la pulsantiera, per esempio, funziona appena si accende la macchina, quando in memoria non c'è ancora alcun programma. Inoltre, nessuno ci impedisce di premere i tasti *mentre* sta girando il programma, cambiando così il contenuto della memoria in modi molto difficili da controllare (in pratica si settano i bit della riga che in quel momento il programma stava leggendo o scrivendo). In un calcolatore moderno questo non è possibile: tutto deve essere sotto il controllo del programma, compresi di dispositivi di I/O.

1.4 La CPU (senza interruzioni e protezione)

Se escludiamo i meccanismi avanzati delle interruzioni e della protezione, quello che resta è funzionalmente molto simile alla CPU del Baby. In particolare, la CPU è un sistema che sa eseguire una sequenza di *istruzioni*. Le istruzioni operano sullo *stato* della CPU e/o interagiscono con l'esterno ordinando operazioni di lettura o scrittura sul bus. Lo stato della CPU è contenuto in un insieme di registri. I registri sono funzionalmente simili alle celle della memoria, con la differenza che si trovano all'interno della CPU. Le istruzioni sono codificate in numeri e contenute in memoria. Uno dei registri della CPU contiene l'indirizzo della prossima istruzione da eseguire (Instruction Pointer, IP). La CPU, da quando la accendiamo a quando la spegniamo, fa solo ed esclusivamente le seguenti cose:

[1] L'I/O è anche collegato direttamente alla CPU, ma possiamo tralasciare questo ulteriore dettaglio.

1. preleva dalla memoria l'istruzione puntata dall'IP;
2. la decodifica, la esegue e aggiorna l'IP;
3. torna al punto 1.

Il modo in cui viene aggiornato l'IP al punto 2 dipende dall'istruzione eseguita. Per la maggior parte delle istruzioni, l'IP viene semplicemente incrementato in modo che punti all'istruzione che segue in memoria, secondo l'ordine degli indirizzi. Un programma, dunque, è per lo più una sequenza di azioni da eseguire una dopo l'altra (come la parola "programma" normalmente indica in altri contesti). Alcune istruzioni possono cambiare l'IP per eseguire dei salti. L'istruzione `hlt` ferma il processore (possiamo pensare che non modifichi l'IP).

Punti (tutti già noti) da tenere a mente:

1. tutto ciò che il processore fa, lo fa perché sta prelevando o eseguendo una istruzione;
2. tranne che quando esegue `hlt`, il processore non smette mai di eseguire istruzioni;
3. le istruzioni sono quelle del linguaggio macchina del processore; il processore non è in grado di eseguire nient'altro;
4. tutto quello che vede il processore è il suo stato corrente e l'istruzione da eseguire; il processore non ricorda le istruzioni passate e non si aspetta particolari istruzioni future:
   - tutto ciò che resta di una istruzione alla fine della sua esecuzione è l'effetto che essa ha avuto sullo stato dei registri del processore, delle celle di memoria, o sui registri di I/O;
   - dualmente, tutto ciò che una istruzione vede quando comincia la sua esecuzione è ciò che si trova nei registri del processore, nelle celle di memoria o nei registri di I/O.

1.5 La memoria ROM

Il fatto che, in un calcolatore moderno, l'ingresso/uscita non funziona direttamente ma deve essere comandato da un programma, crea un problema di *bootstrap*: come facciamo a caricare un programma in memoria se l'unica via di ingresso ha essa stessa bisogno di che ci sia un programma in memoria per poter funzionare? La soluzione è di avere una memoria non volatile che contenga già un programma, fisso, che ha lo scopo di caricare il programma vero e proprio da qualche periferica di I/O (hard disk, rete, DVD, etc.).

1.6 Il flusso di controllo

Quanto esposto sopra è tutto ciò che l'hardware sa fare. Tutto ciò che un calcolatore fa, per quanto complicato possa apparire, deve ridursi a questo. È il software (il programma contenuto in memoria ed eseguito dalla CPU) a orchestrare questi comportamenti elementari in modo da ottenerne di più complessi.

Tutta l'architettura esiste solo per eseguire il software, ed è il software che comanda. La CPU è sotto il controllo del software: la CPU deve eseguire l'istruzione corrente (quella il cui indirizzo si trova nel suo istruction pointer), ed è inoltre l'istruzione stessa a dire quale deve essere l'istruzione successiva. Le istruzioni che non sono di salto lo dicono implicitamente (l'istruzione successiva è la prossima in memoria), mentre quelle di salto lo dicono esplicitamente. L'unico altro meccanismo che dice al processore qual è l'istruzione successiva è quello delle interruzioni, che per il momento ignoriamo.

Il controllo "fluisce" dunque da una istruzione all'altra come deciso dal software stesso. Se il software è composto di più parti, come per esempio più subroutine o diverse applicazioni o librerie, è sempre una sola di queste parti per volta che ha il controllo del processore. Il controllo può essere solo "palleggiato": quando una subroutine S_1 ne invoca un'altra S_2 (per es. tramite una istruzione **`call`**) si dice che S_1 *trasferisce il controllo* a S_2. A questo punto il processore obbedisce a S_2 e non più a S_1, fino a quando S_2 non deciderà di *restituire il controllo* a S_1 (per es. eseguendo una istruzione **`ret`**).

1.7 Hardware e software

Alcuni trovano difficoltà nel capire se una certa cosa è fatta in hardware o in software, e in generale a capire "chi" svolge determinate azioni.

Per orientarsi può essere utile tenere sempre a mente cosa i vari componenti *sanno*. In particolare, si rifletta sul fatto che la CPU sa solo ciò che è contenuto nei suoi registri e, in particolare, vede del programma solo l'istruzione che di volta in volta sta eseguendo, senza ricordare le istruzioni passate e senza guardare quali siano le istruzioni future (che pure sono già scritte in memoria). Questo limita fortemente le cose che la CPU può fare. Nello scenario di S_1 che trasferisce il controllo a S_2, per esempio, la CPU non può controllare che S_2 ritorni a S_1: non sa cosa S_2 stia facendo e, con la visione limitata di una istruzione alla volta, non è in grado di capirlo. La CPU di cui ci occupiamo noi, per di più, non sa nemmeno che nel passato era stata eseguita una **`call`** e che dunque prima o poi deve essere eseguita una **`ret`**.[2]

Non si deve però giungere alla conclusione che dunque tutto è fatto in software. Quando si ritiene che una certa operazione x sia fatta in software, si deve essere in grado di rispondere alle seguenti domande:

[2] Verso la fine del corso vedremo una CPU più intelligente che cerca di ricordare e prevedere più cose, ma lo fa al solo scopo di andare più veloce, restando per il resto equivalente a quella stupida.

- se x è fatta in software, vuol dire che da qualche parte in memoria ci deve essere un programma p (anche di una sola istruzione) che fa x; so scrivere questo programma?
- il software non ha effetto se la CPU non lo esegue; come fa il flusso di controllo a passare da p quando serve che sia fatta x?

Se anche una di queste domande risulta assurda, è probabile che x non sia fatta in software.