Appunti di Sistemi di Elaborazione
Varie sui calcolatori elettronici

MODELLO DI MACCHINA MULTILIVELLO 2
Introduzione *2*
Linguaggi, livelli e macchine virtuali *3*
La struttura a livelli delle macchine odierne *3*
Evoluzione delle macchine a più livelli *5*
L'ORGANIZZAZIONE DEI SISTEMI DI ELABORAZIONE 6
I processori *6*
Organizzazione della ALU *6*
Gli indirizzi di memoria *7*
Ordinamento dei byte 7
Input e Output *7*
Accesso diretto alla memoria *9*
CONTROLLO DI FLUSSO DI UN PROGRAMMA 11
Premessa *11*
Istruzioni per la chiamata di procedure *11*
Le coroutine *11*
I trap e gli interrupt *13*
Controller di interrupt 14
La gestione degli interrupt 14
IL BUS 15
I collegamenti della CPU con il mondo esterno *15*
Amplificazione dei segnali per il bus *16*
Arbitraggio del bus *17*

Modello di macchina multilivello

Introduzione

Possiamo sommariamente dire che un **calcolatore digitale** è una macchina che è in grado di risolvere dei *problemi* eseguendo le *istruzioni* che gli vengono fornite. Una sequenza di istruzioni che descrive come eseguire un certo compito è detta **programma**. Un calcolatore digitale è dotato di una serie di **circuiti elettronici**: questi sono in grado di riconoscere ed eseguire "direttamente" solo un insieme limitato di **istruzioni semplici**; perciò, tutti i programmi devono essere convertiti (ed è sempre possibile farlo) in sequenze di tali istruzioni semplici. L'insieme delle istruzioni semplici di un calcolatore forma un linguaggio che permette di comunicare con il calcolatore. Questo linguaggio prende il nome di **linguaggio macchina**.

Nel progettare un calcolatore, in base all'uso che se ne intende fare, si tende a semplificare il più possibile le istruzioni del linguaggio macchina. Tuttavia, lo svantaggio di questa elevata semplicità sta nella difficoltà, per l'utente, di utilizzarlo. Il problema viene allora risolto progettando un nuovo insieme di istruzioni che risulti più conveniente da usare. Mentre il linguaggio macchina verrà indicato nel seguito con **L1** (e viene spesso detto **linguaggio built-in**), questo secondo insieme di istruzioni verrà indicato come linguaggio **L2**.

Tuttavia, del linguaggio L2 si possono fare 2 usi differenti; infatti, dato che un calcolatore può eseguire sempre e soltanto istruzioni in L1, si tratta di vedere in quali modi viene effettuato il *passaggio* da L2 ad L1:

- il primo metodo consiste nel sostituire ad ogni istruzione in L2 l'equivalente sequenza di istruzioni in L1: con questo metodo, detto di **traduzione**, il calcolatore esegue il nuovo programma in L1 anziché il vecchio in L2;
- il secondo metodo è invece un po’ più complesso: bisogna infatti scrivere scrive un programma, in linguaggio L1, che riceva in input i programmi in L2 e li esegua esaminando una istruzione dopo l'altra ed eseguendo per ognuna la sequenza equivalente di istruzioni in L1. Questo secondo metodo è detto di **interpretazione** ed il programma in L1 che lo realizza prende il nome di **interprete**.

In generale, quindi, *con la traduzione si passa prima al programma equivalente in L1 e poi lo si esegue; al contrario, con l'interpretazione, non viene generato alcun programma intermedio: ogni istruzione in L2 viene esaminata, decodificata e immediatamente eseguita*. La caratteristica comune è, ovviamente, il fatto per cui il calcolatore esegue solo istruzioni in L1.

Ovviamente, per rendere pratici i processi di traduzione e/o di traduzione, è necessario che i linguaggi L1 ed L2 non differiscano troppo: tuttavia, questo fa’ sì che anche L2 risulti non particolarmente comodo da usare per l'utente. Di qui l'idea e spesso la necessità di inventare anche un terzo insieme di istruzioni, questa volta più orientato all'utente che non alla

macchina di quanto non avvenisse per L2. Indicheremo questo linguaggio con **L3**.

Linguaggi, livelli e macchine virtuali

Ogni macchina è dotata di un proprio linguaggio macchina (L1), cioè l'insieme di tutte le istruzioni che essa può eseguire "direttamente". In questo senso possiamo dire che *una macchina definisce un linguaggio*. Vale però anche il viceversa, ossia ogni linguaggio definisce una macchina, intesa come quella macchina che può eseguire direttamente tutti i programmi scritti in quel linguaggio. Chiaramente, a linguaggi complessi (quelli ad alto livello, per esempio) corrisponderanno macchine complesse e costose, ma questo non impedisce di pensare che tali macchine possano comunque esistere.

Possiamo allora visualizzare un calcolatore dotato di **N livelli** (cioè di N linguaggi) come N differenti **macchine VIRTUALI**, ognuna relativa ad un livello. Nel seguito, useremo i termini "livello" e "macchina virtuale" in modo intercambiabile.

Solo quei programmi scritti in linguaggio L1 potranno essere eseguiti direttamente dai circuiti elettronici di una macchina. Ogni programma che sia scritto in un linguaggio diverso da L1 dovrà subire o una serie di *traduzioni* fino ad arrivare ad un programma in L1 oppure dovrà essere *interpretato* da un interprete di livello inferiore, anche qui fino ad uno di livello L1. Chiaramente, però, una persona che scrive programmi per la macchina virtuale di livello N non si deve preoccupare degli interpreti e dei traduttori sottostanti: La struttura della macchina garantisce che questi programmi saranno in qualche modo eseguiti.

La struttura a livelli delle macchine odierne

Nei calcolatori moderni si possono generalmente individuare i seguenti livelli:

- il livello più basso (**livello 0**) è quello dell' **hardware**, ossia dei circuiti elettrici ed elettronici di cui il calcolatore si compone. Tali dispositivi eseguono direttamente i programmi scritti nel linguaggio macchina del livello superiore, cioè il livello 1 ([1]);.
- il livello successivo (**livello 1**) è il livello del vero **linguaggio macchina**: è a partire da questo livello che si introduce il concetto di *programma* come sequenza di istruzioni da eseguire; in particolare, nel livello 1 c'è un programma, detto **microprogramma**, che si occupa di interpretare le istruzioni del livello 2. Il livello 1 viene detto **livello della microprogrammazione**. Da notare che, per definizione stessa di tale

[1] Ci sarebbe in verità anche un livello più basso del livello 0, il cosiddetto **livello di dispositivo** che però riguarda il campo della ingegneria elettronica e non ci interessa. Il livello 0 è conosciuto come **livello della logica digitale**.

livello, non possono esistere 2 calcolatori aventi lo stesso livello 1: ci possono essere, e ci sono, similitudini, ma mai l'uguaglianza;

- il **livello 2** è noto come **livello della macchina standard**. Bisogna dire che non tutti i calcolatori dispongono del livello 1: in altre parole, in essi i programmi di livello 2 vengono eseguite dai circuiti di livello 0 senza traduzioni o interpretazioni intermedie;
- al di sopra del livello 2 c'è il **livello 3** che presenta alcune particolari caratteristiche: intanto, esso di differenzia dal livello 2 in quanto fornisce un insieme di nuove istruzioni, una organizzazione della memoria differente, la capacità di eseguire più programmi in parallelo ed altro; tuttavia, gran parte delle istruzioni del livello 3 compaiono anche nel livello 2. Ecco perchè potremmo dire che si tratta di un **livello ibrido**. Le nuove capacità aggiunte a livello 3 vengono eseguite da un particolare interprete il quale fa operare il livello 2: si tratta del **sistema operativo**. In altre parole, le istruzioni del livello 3 identiche a quelle del livello 2 vengono eseguite dal livello 2, ossia vengono interpretate direttamente dal microprogramma (livello 1) corrispondente; le altre istruzioni vengono invece interpretate dal sistema operativo. Il livello 3 è noto come **livello del sistema operativo**;
- mentre tra i livelli finora esaminati ci sono fondamentalmente notevoli affinità, il passaggio dal livello 3 al **livello 4** è molto più brusco e netto. Lo scopo dei primi livelli (da 0 a 3) non è certo un utilizzo diretto da parte del programmatore medio, bensì il funzionamento dei *traduttori* e degli *interpreti* che supportano i livelli superiori. Al contrario, tali livelli superiori sono concepiti per un uso diretto da parte del programmatore, il quale li utilizza per la risoluzione dei propri problemi. La differenza forse più evidente è la seguente: mentre i linguaggi dei livelli 1, 2 e 3 sono costituiti solo da numeri (ottimi per le macchine ma pessimi per le persone), i linguaggi dal livello 4 in su cominciano a contenere *stringhe* ed *abbreviazioni* sicuramente più congeniali per l'uomo. Il livello 4 è noto come **livello del linguaggio Assembler**: si tratta fondamentalmente di una *forma simbolica* di uno dei linguaggi sottostanti. In pratica, questo livello consente di scrivere programmi formalmente analoghi a quelli dei livelli inferiori, ma comunque meno ostici. I programmi del livello 4 vengono prima tradotti nei linguaggi di livello 1, 2 e 3 e poi interpretati per l'esecuzione. I programmi dediti alla traduzione sono gli **assemblatori**;
- al di sopra del livello 4 c'è il **livello dei linguaggi simbolici ad alto livello**, ossia di linguaggi molto vicini al linguaggio naturale dell'uomo. Programmi scritti in questi linguaggi vengono solitamente tradotti in programmi di livello 4 o 3 dai cosiddetti **compilatori**.

Evoluzione delle macchine a più livelli

Esaminiamo velocemente lo sviluppo storico delle macchine a più livelli.

I primi **calcolatori digitali** (1940) possedevano soltanto 2 livelli: il livello di macchina standard, in cui era fatta tutta la programmazione, e quello della logica digitale, che eseguiva i programmi.

Nel 1951 si passò a calcolatori a 3 livelli, al fine soprattutto di semplificare l'hardware: ogni macchina aveva adesso un interprete la cui funzione era quella di eseguire i programmi del linguaggio della macchina standard tramite la loro interpretazione.

Nel giro di pochi anni si passò però al livello degli assemblatori e dei compilatori, allo scopo di facilitare il compito dei programmatori.

Agli inizi, i calcolatori erano un qualcosa su cui i programmatori potevano e dovevano operare personalmente per far funzionare i propri programmi. Questo implicava che essi dovessero conoscere a fondo la macchina e la soluzione di eventuali problemi. Conseguenza: era più il tempo in cui si cercava di individuare i guasti che non il tempo dedicato alle esecuzioni dei programmi.

Dopo 10 anni (1960), si tentò di ridurre questa perdita di tempo automatizzando il lavoro degli operatori: venne introdotto un programma, detto **sistema operativo**, il quale, una volta ricevuti i programmi in input dal programmatore, si occupava di leggerli ed eseguirli. La sofisticazione dei sistemi operativi fu molto rapida: nuove istruzioni, opzioni e caratteristiche furono aggiunte al livello della macchina standard, finché non costituirono un nuovo livello, quello appunto del sistema operativo.

I primi sistemi operativi non facevano altro che leggere i *pacchi di schede* forniti dai programmatori e stampavano gli output su una stampante: questo tipo di organizzazione era conosciuta come sistema **bacth** o **a lotti** ed era generalmente piuttosto lenta. Più tardi, invece, furono invece sviluppate delle versioni dei sistemi operativi che permettevano a diversi programmatori di comunicare direttamente con il calcolatore. In questo tipo di sistemi, venivano usati dei cavi telefonici per collegare i terminali al calcolatore centrale. In questo modo, il programmatore poteva inviare i propri programmi ed osservare i risultati (in tempi molto rapidi) da qualunque luogo. Questi sistemi furono e sono tuttora chiamati **sistemi time-sharing**.

L'ORGANIZZAZIONE DEI SISTEMI DI ELABORAZIONE

I processori

L'organizzazione di un semplice calcolatore dotato di **BUS** per il collegamento tra i dispositivi è il seguente:

- il cuore del sistema è costituito dalla **CPU** (*Unità centrale di elaborazione*), la quale consta delle seguenti componenti: a) **unità di controllo** b) **unità aritmetica e logica** (**ALU**) c) **registri interni**;
- in posizione fisicamente adiacente alla CPU c'è la **memoria centrale** (di tipo **RAM**, Random Access Memory) del calcolatore;
- seguono i vari **dispositivi di Input** (Disco, tastiera,...) e di **Output** (Video, stampante, ...).

Occupiamoci in particolare della CPU e delle sue componenti:

- l' **unità di controllo** ha il compito di prelevare le istruzioni dalla memoria principale e di determinarne il tipo;
- l' **unità aritmetico-logica** esegue le operazioni booleane elementari (OR, AND,..) che corrispondono alle istruzioni;
- i **registri interni** sono una specie di piccola memoria ad altissima velocità usata per memorizzare i *risultati temporanei* e certe *informazioni di controllo*; ognuno di questi registri ha una particolare funzione: il registro più importante è il **Program Counter**, il quale punta alla prossima istruzione da eseguire; segue il registro delle istruzioni **Instruction Register**, il quale contiene di volta in volta l'istruzione da eseguire.

Organizzazione della ALU

Quando l'unità aritmetica e logica deve eseguire una operazione tra 2 dati, il "percorso" di tali dati, secondo lo schema di **Von Neumann**, è il seguente: i due dati vengono prelevati dai registri interni della CPU oppure direttamente dalla memoria; vengono quindi inviati a 2 registri di input per l'ALU; l'ALU esegue l'operazione richiesta sui dati contenuti in tali 2 registri e deposita il risultato in un registro di output (sempre specifico della ALU) dal quale i dati verranno successivamente trasportati nella destinazione finale.

Gli indirizzi di memoria

La memoria principale del calcolatore è costituita da un numero di **celle** (o **locazioni**), ognuna delle quali può immagazzinare un *elemento di informazione*. Ogni cella è individuata da un proprio numero (detto **indirizzo**) con il quale i programmi possono riferirsi ad essa. Se una memoria ha N celle, allora tali celle avranno indirizzi compresi tra 0 e N-1. Ogni cella di memoria contiene lo stesso numero di bit. Se una celle contiene K bit, allora essa può contenere una qualsiasi delle 2^K combinazioni diverse di tali bit. Le celle adiacenti hanno ovviamente indirizzi consecutivi.

I calcolatori che usano l' **aritmetica binaria** esprimono anche gli indirizzi di memoria tramite numeri binari. Ad esempio, se un indirizzo di memoria binario contiene M bit, allora significa che il numero massimo di celle direttamente indirizzabili è 2^M. Il numero di bit nell'indirizzo è collegato al numero massimo di celle direttamente indirizzabili mentre invece non dipende affatto dal numero di bit per ogni cella.

Ordinamento dei byte

I byte che costituiscono una **parola** possono essere numerati da sinistra verso destra o viceversa. Ad esempio, mentre nella famiglia **Motorola** (a 32 bit) i byte sono normalmente numerati da sinistra a destra, al contrario, nella famiglia **INTEL** (a 32 bit) avviene il contrario, ossia la numerazione è da destra verso sinistra. Nel primo caso di parla di **calcolatori a finale grande**, mentre nel secondo caso di parla di **calcolatori a finale piccolo**.

Naturalmente, però, i numeri vengono memorizzati sempre da destra verso sinistra: per esempio, supponiamo di avere il numero intero 6, il quale, espresso in binario con 32 bit, è

```
00000000 00000000 00000000 00000110
```

Nel caso del sistema a finale grande (Motorola), questa configurazione viene inserita, così com'è, a partire dal byte 0 per finire al byte 3. Al contrario, nel caso del sistema a finale piccolo (INTEL), il primo byte a sinistra va nel byte 3, il secondo a sinistra va nel byte 2, il successivo nel byte 1 e l'ultimo, quello che contiene i bit significativi, va nel byte 0. In entrambi i casi, però, l'indirizzo della parola che contiene questo numero è SEMPRE zero. In un certo senso, cambia l'ordine con cui i bit vengono "letti": mentre nel finale grande si legge a partire dalla cella avente l'indirizzo specificato e si va per indirizzi crescenti, nel finale piccolo si fa' il contrario.

Input e Output

Nei moderni calcolatori si usano solitamente due diverse organizzazione del sistema di I/O. Consideriamo ad esempio i **Main Frame**, i quali usano uno schema di questo tipo: intanto, il sistema di calcolo della macchina è costituito dalla CPU (che può essere affiancata anche da altre CPU), dalla

memoria centrale e da un o più altri "processori specializzati" per l'input/output e che vengono chiamati **canali dei dati**. Tutti i dispositivi di I/O sono collegati a questi canali. Quando la CPU necessita di una operazione di I/O, il meccanismo che viene utilizzato è il seguente:

- una volta individuato il dispositivo di I/O al quale del quale si vuol fare uso, la CPU carica nel relativo canale un programma speciale e dice al canale stesso di eseguirlo;
- il canale, quindi, si occupa di manipolare tutti gli I/O che sono diretti alla memoria centrale (input) o che provengono dalla memoria centrale (output), in modo da lasciare libera la CPU di compiere altre operazioni;
- appena terminato il proprio compito, il canale spedisce alla CPU un segnale, detto **interruzione**, avvisando di aver terminato i propri compiti e richiedendo la sua attenzione.

Il vantaggio di un meccanismo del genere è evidente: tutto il lavoro di I/O viene demandato al canale, mentre la CPU può occuparsi d'altro; in questo modo, i calcoli e l'I/O possono avvenire contemporaneamente.

I main frame, che manipolano generalmente grandi quantità di dati per l'I/O, contengono 3 distinti BUS: un primo bus collega direttamente ogni canale alla memoria centrale, in modo che il canale stesso possa leggere e scrivere autonomamente parole di memoria; un secondo bus collega la CPU a ciascun canale, in modo che essa possa inviare i comandi ai canali e che ogni canale possa inviare alla CPU i segnali di interruzione; infine c'è il bus che collega direttamente la CPU alla memoria centrale.

Nel caso invece dei **personal computer**, l'organizzazione è evidentemente più semplice; per queste macchine è prevista una grande piastra a circuito stampato, detta **scheda madre**: essa contiene i chip della CPU, un po' di memoria, alcuni chip di supporto ed un unico bus inciso lungo la sua lunghezza; tale bus presenta dei *connettori* ai quali possono essere inseriti i *connettori terminali* sia della memoria addizionale sia delle schede dei dispositivi di I/O.

Lo schema di un PC è perciò di questo tipo: c'è un unico bus che collega la CPU con la memoria e con i dispositivi di I/O; in particolare, ogni dispositivo di I/O è costituito da due parti: la prima parte, quella direttamente collegata al bus, contiene la maggior parte della elettronica chiamata **controllore**; la seconda contiene il dispositivo fisico vero e proprio. Il controllore è contenuto in una scheda che, come detto prima, viene inserita sulla piastra madre (tramite opportuni **slot**) e viene collegata al bus. Solo alcuni controllori (ad esempio quello della tastiera) vengono direttamente collocati sulla scheda madre, senza l'uso di piastre aggiuntive.

Il compito del controllore è quello di controllare il suo dispositivo di I/O e di gestire l'accesso di tale dispositivo al BUS. Supponiamo, ad esempio, che il programma in esecuzione in un certo momento richieda dei dati dal **disco fisso**; il meccanismo che parte è il seguente:

- il programma stesso dà un comando al controllore del disco;
- il controllore dà a sua volta dei comandi al drive e comincia la ricerca dei dati richiesti;
- una volta localizzati la pista ed il settore appropriati, il drive comincia a far pervenire i dati al controllore, tramite un flusso di bit;
- il controllore suddivide tale flusso di bit in parole e, tramite il bus, scrive ogni parola in memoria.

In particolare, ci sono dei controllori che sono in grado di leggere o scrivere parole in memoria senza che alcun intervento "vigile" della CPU: in questo caso si dice che il controllore in questione ha un **accesso diretto alla memoria** (**DMA**).

Il bus non è ovviamene utilizzato solo dai controllori di I/O; anche la CPU usa il bus per prelevare dati e istruzioni. Può accadere allora che, nello stesso istante, la CPU ed un controllore richiedano l'accesso al bus. A stabilire chi dei due deve avere per primo l'accesso al bus è uno specifico chip, detto **controllore del bus**: in generale, esso dà la precedenza ai dispositivi di I/O per motivi di sicurezza dei dati. Ovviamente, quando nessun dispositivo di I/O richiede l'accesso al bus, allora la CPU ha l'accesso completo. Il processo per cui i dispositivi di I/O vengono sempre privilegiati, rispetto alla CPU, per l'accesso al bus prende il nome di **furto di cicli** ed è un fattore di rallentamento del calcolatore.

Accesso diretto alla memoria

Parliamo un po' più diffusamente dell' **accesso diretto alla memoria**. Intanto, il **chip DMA**, ossia il "chip controllore del DMA" che si occupa di gestire l'accesso diretto dei dispositivi di I/O alla memoria, dispone di almeno 4 registri: il primo contiene l'indirizzo di memoria del byte da leggere o scrivere; il secondo contiene il numero di byte da trasferire; il terzo specifica il numero del dispositivo oppure l'indirizzo dello spazio di I/O da usare; infine, il quarto dice se i dati sono da leggere o da scrivere da o in un dispositivo di I/O.

Una volta che questi 4 registri sono stati inizializzati per il compimento dell'operazione di I/O richiesta, il controllore DMA fa una *richiesta di accesso al bus* per poter accedere alla memoria all'indirizzo specificato dal primo registro: tale richiesta è del tutto analoga a quella che viene compiuta dalla CPU quando intende accedere anch'essa alla memoria. Una volta ottenuto l'accesso al byte richiesto, il controllore DMA fa una richiesta al dispositivo di I/O, il cui numero è indicato nel terzo registro, per poter o ricevere il byte da scrivere oppure inviare il byte da leggere. Fatto questo, il controllore del DMA incrementa il registro di indirizzo (il primo) di una unità mentre decrementa il registro contatore (il secondo) della stessa quantità. A questo punto, nel caso che il registro contatore sia diverso da zero, l'operazione si ripete per il nuovo indirizzo; in caso contrario, il controllore DMA ferma il trasferimento dei dati e invia l'interrupt alla CPU per segnalare che l'operazione è stata compiuta.

E' evidente che, con il metodo DMA, la CPU deve semplicemente inizializzare pochi registri, dopodiché può occuparsi d'altro finché il controllore di DMA non termina i suoi compiti. Da notare, tuttavia, che il metodo del DMA, pur alleggerendo il lavoro della CPU per l'I/O, non velocizza poi molto l'elaborazione: infatti, la ridotta velocità dei dispositivi di I/O rispetto alla CPU rende necessari vari cicli del bus sia per gli accessi alla memoria sia per quelli ai singoli dispositivi. Quindi, durante questi cicli, dovendo dare la precedenza al DMA, la CPU viene spesso costretta ad attendere. Di qui il nome di "furto di cicli" di cui si è parlato prima. Ad ogni modo, la tecnica del DMA è utilizzata oggi in quasi tutti i PC ed i minicalcolatori.

La situazione è invece diversa nei main frame, dei quali abbiamo comunque già parlato prima. In queste macchine, dove l'I/O è enorme, non è possibile tollerare il furto dei cicli perchè esso saturerebbe il bus. Ecco allora che vengono utilizzati dei chip speciali che abbiamo definito "canali dei dati". Un canale è in realtà un calcolatore specializzato: gli può essere affidato un programma che esso esegue senza alcun intervento da parte della CPU centrale.

Come riferimento pratico per descrivere il funzionamento dei canali consideriamo un main frame della IBM. Ci sono 2 diversi tipi di canali: un **canale selettore** si occupa di gestire i dispositivi ad alta velocità, quali i dischi; proprio a causa dell'alta velocità, un canale di questo tipo può gestire 1 solo dispositivo per volta; al contrario, un **canale multiplexer** può gestire contemporaneamente molti dispositivi a bassa velocità.

Vediamo allora cosa succede quando in un main frame si deve eseguire un I/O:

- la CPU crea prima un programma per il canale e lo memorizza nella memoria centrale;
- quindi essa esegue una istruzione "Start I/O" specificando sia il canale sia il dispositivo di cui si richiede l'utilizzo;
- a questo punto, il canale preleva l'indirizzo del programma che deve eseguire: tale indirizzo si trova in una locazione fissa predefinita della memoria ed il canale lo pone nel proprio registro contatore di programma;
- quindi comincia ad eseguire il suddetto programma;
- alla fine della esecuzione, che corrisponde quindi alla fine dell'operazione di I/O, viene inviato un interrupt alla CPU.

CONTROLLO DI FLUSSO DI UN PROGRAMMA

Premessa

Il **flusso di controllo** si riferisce alla sequenza in cui le istruzioni di un programma vengono eseguite. Nel caso più semplice, le istruzioni vengono prelevate da *locazioni di memoria consecutive*. Alterazioni a tale ordine sequenziale vengono da istruzioni di chiamata di procedura (in generale da **istruzioni di salto**), in quanto fermano l'esecuzione della procedura in corso e fanno cominciare la procedura chiamata, e da altri tipi di istruzioni che esamineremo nel dettaglio nei prossimi paragrafi.

Istruzioni per la chiamata di procedure

Una **procedura** è un gruppo di istruzioni che esegue un certo compito e che può essere chiamato da un qualsiasi punto del programma. Quando la procedura ha finito il suo compito, deve ritornare alla istruzione che segue la chiamata: questo implica che l' **indirizzo di ritorno** debba essere in qualche modo trasmesso ala procedura. Ci sono 3 posti in cui l'indirizzo di ritorno può essere sistemato: la memoria, un registro o lo **stack**. Di queste però, la soluzione peggiore è l'uso di una locazione fissa di memoria: infatti, ciò escluderebbe la possibilità di effettuare una seconda chiamata, in quanto ogni chiamata distruggerebbe l'indirizzo di ritorno della chiamata precedente. Migliore è l'uso di un registro, nel senso che l'istruzione di chiamata pone in tale registro l'indirizzo di ritorno. Tuttavia, in questo modo, per chiamate successive si dovrebbero occupare più registri e la cosa non è ovviamente possibile. La soluzione migliore è senz'altro l'uso dello stack: ogni istruzione di chiamata spinge (**push**) l'indirizzo di ritorno nello stack; quando la procedura è finita, estrae (**pop**) ed elimina l'indirizzo di ritorno dallo stack e lo mette nel contatore di programma.

Le coroutine

Quando c'è una istruzione di chiamata di procedura, c'è una evidente quanto fondamentale distinzione tra la **procedura chiamante** e la **procedura chiamata**. Supponiamo che ci sia una procedura A che, ad un certo punto della proprie esecuzione, chiama la procedura B:

- tale procedura (a meno di ulteriori chiamate) lavora per un certo tempo (quello necessario alla esecuzione delle sue istruzioni) e poi ritorna alla procedura A;
- per chiamare B, A usa una *istruzione di chiamata di procedura* la quale pone l'indirizzo di ritorno (che è l'indirizzo dell'istruzione seguente la chiamata) sulla cima dello stack e l'indirizzo di B (cioè della prima istruzione di B) nel contatore di programma;

- una volta terminata, B usa una *istruzione di ritorno*, la quale preleva l'indirizzo di ritorno dallo stack e ponendolo nel contatore di programma.

La cosa fondamentale da notare in questi passaggi è la seguente: la procedura chiamata (cioè B) viene eseguita, dopo ogni chiamata, sempre dall'inizio alla fine; la procedura chiamante (cioè A), una volta eseguita B, riprende non dall'inizio ma dall'istruzione seguente la chiamata.

Una cosa diversa accade quando invece 2 procedure A e B si chiamano a vicenda: per esempio, supponiamo che in ciascuna di queste 2 procedure ci siano varie chiamate all'altra; supponiamo che per prima venga chiamata A (per esempio dal programma principale); all'interno di A viene chiamata B, la quale quindi comincia dall'inizio; ad un certo punto di B c'è una chiamata ad A: tale chiamata è però tale che l'esecuzione di A non avvenga dall'inizio, bensì dall'istruzione successiva alla chiamata di B; proseguendo, c'è una nuova chiamata a B, la quale riprende l'esecuzione non dall'inizio, ma nel punto in cui aveva poco fa chiamato A e così via.

Due procedure che si considerano a vicenda una procedura (poiché sono chiamate, eseguono un certo lavoro e poi ritornano all'istruzione che segue la chiamata) vengono definite **coroutine**. Da quanto si è detto, si deduce che, per un meccanismo di questo tipo, non bastano le normali istruzioni di chiamata e di ritorno. Per poter realizzare questo meccanismo serve una istruzione che possa semplicemente scambiare la cima dello stack con il contatore di programma, usando per esempio un registro interno come variabile di scambio. Vediamo di analizzare il meccanismo considerando un programma principale e le due procedure A e B:

- il programma principale chiama A: viene memorizzato l'indirizzo di ritorno nello stack e viene inserito nel contatore di programma (*program counter*) l'indirizzo di A;
- ad un certo punto, A chiama B: l'indirizzo di ritorno viene impilato e l'indirizzo di B viene inserito nel contatore di programma;
- B chiama A: vengono scambiati il program counter e la cima dello stack: in tal modo il program counter contiene l'indirizzo della istruzione di A successiva alla chiamata di B, mentre la cima dello stack contiene l'indirizzo dell'istruzione di B successiva all'ultima chiamata;
- ad ogni chiamata lo stack non viene né incrementato né decrementato;
- alla fine delle istruzioni di B, si passa ad eseguire le ultime istruzioni di A con lo stack che contiene, in cima, l'indirizzo di ritorno al programma principale.

Una istruzione di chiamata di coroutine viene spesso detta **istruzione di resume** ed è presente nelle macchine di livello 2 (livello del linguaggio Assembler).

I trap e gli interrupt

Un **trap** è semplicemente un tipo di chiamata di procedura: si tratta però di una chiamata automatica attivata da una certa condizione provocata dal programma (ad esempio l'overflow). Quando si verifica una certa condizione (piuttosto rara per la verità) durante l'esecuzione di un programma, interviene un trap, il quale devia il flusso di controllo verso una locazione di memoria fissata; in tale locazione c'è un salto ad una opportuna procedura chiamata **gestore di trap**.

Anche gli **interrupt** sono cambiamenti nel flusso di controllo durante l'esecuzione del programma: tuttavia, essi non dipendono dall'esecuzione del programma, bensì da qualcos'altro (spesso sono in relazione a operazioni di I/O). Al pari dei trap, un interrupt ferma il programma in corso e trasferisce il controllo ad un **gestore di interrupt**, il quale esegue le azioni appropriate. Quando ha finito, il gestore di interrupt restituisce il controllo al programma interrotto. In particolare, il programma deve riprendere dall'esatto punto in cui era stato sospeso, il che significa che è necessario ripristinare tutti i registri interni allo stato di pre-interruzione.

I moderni calcolatori sono spesso sotto il controllo di un **insieme gerarchico di programmi**, alla cui sommità c'è naturalmente il sistema operativo. Per poter trasferire il controllo del calcolatore da un certo programma ad un altro (a seconda della operazione che si intende fare) è necessario fornire un meccanismo mediante il quale il funzionamento di un programma possa essere temporaneamente "interrotto" a favore di un altro. Una interruzione, come si detto poco fa, è un evento oppure un preciso segnale esterno, che provoca appunto la sospensione del funzionamento di un programma. Quando si verifica una interruzione, il contenuto di tutti i registri viene immediatamente salvato e il controllo del calcolatore passa ad una **routine di servizio di interruzione** facente parte, in genere, del sistema operativo. Essa determina da dove viene la richiesta di interruzione (generalmente esaminando un certo numero di flag o indicatori) e quindi trasferisce a sua volta il controllo ad una routine per la gestione di quel particolare tipo di interruzione. Successivamente, il funzionamento del programma viene ripreso esattamente dal punto in cui era stato sospeso: questo lo si ottiene facilmente inserendo in tutti i registri i valori che erano stati salvati al momento della interruzione.

Tra le cause più comuni di interruzioni vi sono operazioni di input e output di dati, oppure il rilevamento di un errore in un programma o anche il fatto che un programma ha superato il limite di tempo previsto. Il repertorio di istruzioni della maggior parte degli elaboratori include delle apposite istruzioni che attivano o disattivano le interruzioni. Esempi di **interrupt HARDWARE** sono:

- divisione per zero
- caduta di tensione
- comunicazione eventi hardware al processore
- esaurimento carta stampante
- completamento di una azione di un drive e così via.

Esempi di **interrupt SOFTWARE** sono invece:

- attivazione programmi del BIOS definiti dal sistema operativo
- attivazione programmi del BIOS definiti e gestiti dal software applicativo
- attivazione programmi di tabella (memorizzazione di indirizzi).

Ogni interrupt specifico è identificato da un numero che ne specifica il tipo. Per ogni interrupt è previsto un programma detto **Gestore di Interrupt** che esegue l'intero lavoro richiesto dall'interrupt. L'indirizzo del gestore è contenuto a sua volta nella cosiddetta **tabella degli interrupt**.

Controller di interrupt

Questo dispositivo serve per gestire le priorità dei vari interrupt, nel senso che, quando ci sono contemporaneamente 2 richieste di interrupt al microprocessore, stabilisce quale debba essere servita per prima e quale debba restare in attesa.

La differenza essenziale tra i trap e gli interrupt è che i primi sono **sincroni** mentre gli altri sono **asincroni**. Questo significa che, se un programma viene eseguito 10 volte con lo stesso input, i trap riappariranno allo stesso punto ogni volta, mentre gli interrupt possono variare.

La gestione degli interrupt

Un concetto chiave legato agli interrupt è la cosiddetta **trasparenza**: quando capita un interrupt, vengono avviate determinate azioni ed eseguite determinate procedure, ma, quando tutto questo è stato fatto, il calcolatore dovrebbe ritornare nello stesso stato in cui era prima dell'interrupt. Una routine di interrupt che ha questa proprietà si dice che è **trasparente**.

Mentre sui piccoli calcolatori il processo di gestione di interrupt risulta complessivamente uguale a quello prima descritto, nei grandi calcolatori le cose si complicano per via del fatto che ci sono molti dispositivi di I/O e molti utenti. Infatti, per questi calcolatori spesso accade che, mentre si sta eseguendo una certa routine di interrupt, ci siano una o più altre richieste di interrupt da parte di altri dispositivi. Per uscire da situazioni di questo tipo, si possono adottare 2 diversi meccanismi:

- una prima soluzione è la seguente: quando una routine di interrupt viene chiamata, prima ancora di salvare il contenuto dei registri, essa deve fare in modo che ogni altra possibile richiesta di interrupt venga ignorata; deve cioè "disabilitare" gli interrupt seguenti. In tal modo, gli interrupt vengono eseguiti in ordine sequenziale: una volta terminata una routine, si passa all'interrupt immediatamente successivo, poi al successivo e così via. Tuttavia, questo metodo può comportare dei problemi per quei dispositivi che non possono tollerare grandi ritardi della CPU nel prestare loro attenzione;

- in questi casi, la strategia più conveniente è la seguente: tenendo conto del tempo che i vari dispositivi possono passare in attesa di essere "serviti", si assegna a ciascuno di essi (cioè ai rispettivi interrupt) una **priorità**, che ovviamente sarà alta per quelli critici e più bassa per gli altri. In questo modo, mentre si sta eseguendo una routine di interrupt di priorità N, ogni tentativo di provocare un interrupt fatto da qualsiasi dispositivo con priorità minore di N è ignorato, finché la routine di interrupt non è finita; solo alla fine della suddetta routine, saranno servite le richieste di interrupt, sempre a partire da quelle con priorità maggiore.

Nel caso dei processori INTEL, ci sono **2 livelli di interrupt**, cioè 2 priorità: ci sono gli **interrupt mascherabili** (bassa priorità) e quelli **non mascherabili** (alta priorità). Gli interrupt della seconda categoria sono usati solo per segnalare errori gravi imminenti, mentre, ad esempio, tutti i dispositivi di I/O usano interrupt mascherabili.

Il BUS

I collegamenti della CPU con il mondo esterno

Un tipico microprocessore ha da 40 a 132 **piedini** attraverso i quali avviene la comunicazione con il mondo esterno; alcuni piedini inviano segnali all'esterno, alcuni ne ricevono dall'esterno, altri fanno entrambe le cose. I piedini di un microprocessore si dividono in 3 tipi: **di indirizzo**, **di dati** e **di controllo**.

Tali piedini sono collegati ad altri piedini che si trovano nei chip della memoria e in quelli di I/O: il collegamento avviene mediante una serie di cavi paralleli che costituiscono il cosiddetto **bus**.

Un BUS appare sostanzialmente come un insieme di collegamenti elettrici che corrono parallelamente lungo il computer. Il BUS si compone di vari **sottobus**, ognuno dei quali trasporta qualcosa:

- intanto c'è il **bus di alimentazione**, che porta appunto corrente dall'alimentazione a tutte le componenti del computer; per i chip a 16 bit, tale bus ha solo 2 linee, che sono l'alimentazione e la massa;
- poi c'è il **bus di controllo**, il quale trasporta le informazioni che riguardano la *temporizzazione* (i segnali di clock del sistema), i *comandi* (per la memoria o per l'I/O), la *direzione dei dati* (lettura o scrittura), i *segnali di disponibilità* a trasmettere o ricevere dati, gli *interrupt*;
- quindi c'è il **bus degli indirizzi**, il quale trasporta degli speciali segnali di controllo usati per distinguere le celle di memoria oppure i vari dispositivi di I/O; gli indirizzi vengono trasmessi lungo questo bus in un codice binario, in base al quale ogni conduttore che trasporta un segnale corrisponde ad uno 0 (bassa tensione) oppure ad un 1 (alta tensione); i primi processori a 8 bit avevano 16 linee per il bus indirizzi,

potendo perciò produrre 2^{16} diversi indirizzi; successivamente, la necessità di gestire maggiori spazi di indirizzamento ha portato l'IEEE ad indicare, per il bus S-100, uno standard di 24 linee di segnale per gli indirizzi; nell'8086 e nell'8088 il bus indirizzi era a 20 bit;

- infine c'è il **bus dei dati**, il quale trasporta l'informazione reale; nei processori a 8 bit, le linee a disposizione sono 8; nei chip a 16 bit, invece, il bus dati è generalmente a 16 linee.

Consideriamo, tanto per fare un esempio, il meccanismo di **fetch** (prelevamento) di una istruzione da parte del microprocessore (**µP**):

- il microprocessore mette l'indirizzo di memoria di quella istruzione sui suoi piedini di indirizzo;
- quindi attiva una linea di controllo per informare la memoria che vuole leggere una parola;
- la memoria risponde mettendo la parola richiesta sui piedini dei dati del microprocessore e invia un segnale di assenso per dire che ha esaudito la richiesta;
- il microprocessore, ricevuto questo segnale, accetta la parola; se l'istruzione è completa, la esegue, altrimenti ripete il procedimento per prelevare le parole rimanenti.

Per quanto riguarda i piedini di controllo, li possiamo raggruppare nelle seguenti categorie:

- **controllo del bus**: servono per inviare dalla CPU ai chip di memoria e di I/O, attraverso il BUS, l'intenzione della CPU di leggere o scrivere o di fare qualcos'altro;
- **interrupt**: questi sono segnali diretti che i dispositivi di I/O inviano alla CPU per comunicare o che devono avviare una operazione di trasferimento oppure che l' hanno appena conclusa;
- **arbitraggio del bus**: questi piedini sono necessari per regolare il traffico nel bus e per evitare che due dispositivi cerchino di utilizzare il bus contemporaneamente (da notare che, per gli scopi di arbitraggio, la CPU è considerata come un normale dispositivo, talvolta anche con priorità minore rispetto ad altri dispositivi);
- **segnalatori di coprocessori**: questi piedini vengono usati, in presenza di **coprocessori**, per effettuare e garantire varie richiese;
- **stato**: questi piedini forniscono o accettano informazioni sullo stato del sistema.

Amplificazione dei segnali per il bus

I segnali binari che i dispositivi del calcolatore emettono non sono normalmente abbastanza *forti* per alimentare un bus, specialmente se è

relativamente lungo ed ha anche molti dispositivi innestati. Di conseguenza, tali dispositivi sono spesso collegati al bus mediante ulteriori dispositivi in grado di amplificare sia i segnali in uscita sia quelli in entrata.

Arbitraggio del bus

Che cosa accade quando due o più dispositivi richiedono contemporaneamente l'accesso al bus? E' necessario utilizzare un meccanismo di **arbitraggio del bus** per evitare il caos.

I meccanismi di arbitraggio possono essere centralizzati o meno. Nel caso dell' **arbitraggio centralizzato**, c'è un solo arbitro del bus che determina la successione degli accessi. Talvolta questo dispositivo è integrato nel chip della CPU talvolta ne è separato.

Il funzionamento di questo tipo di arbitraggio è il seguente:

- quando un certo dispositivo ha bisogno di accedere al bus, invia un segnale su una opportuna linea;
- l'arbitro nota la richiesta del bus e invia un segnale di assenso sulla linea di concessione del bus;
- questa linea è collegata ai dispositivi di I/O posti in serie;
- quando il dispositivo FISICAMENTE più vicino all'arbitro vede l'assenso, controlla se ha fatto la richiesta: se sì, accede al bus e non propaga ulteriormente l'assenso; se no, passa l'assenso al dispositivo adiacente.

Il processo termina quando il dispositivo che ha fatto effettivamente richiesta vede l'assenso e accede al bus.

Questo schema viene detto **catena di priorità**: ad ogni dispositivo è assegnata una priorità in funzione della vicinanza fisica all'arbitro del bus.

Si tratta di un meccanismo evidentemente molto semplice; nella maggior parte dei casi, per aggirare le priorità che dipendono dalla distanza dall'arbitro, i bus utilizzano **livelli di priorità multipli**: questo significa che, per ogni livello di priorità, c'è una linea di richiesta ed una di concessione e che ogni dispositivo è collegato ad un livello in funzione della quantità di tempo che può restare in attesa. Se più livelli di priorità vengono richiesti contemporaneamente, l'assenso viene dato al livello maggiore, mentre invece tra i dispositivi con la stessa priorità si usa lo schema della catena di priorità prima illustrato.

Autore: **Sandro Petrizzelli**
e-mail: sandry@iol.it
sito personale: http://users.iol.it/sandry
succursale: http://digilander.iol.it/sandry1