PC inside

come funziona un Sistema di Elaborazione

(2007.05)

Indice

Premessa

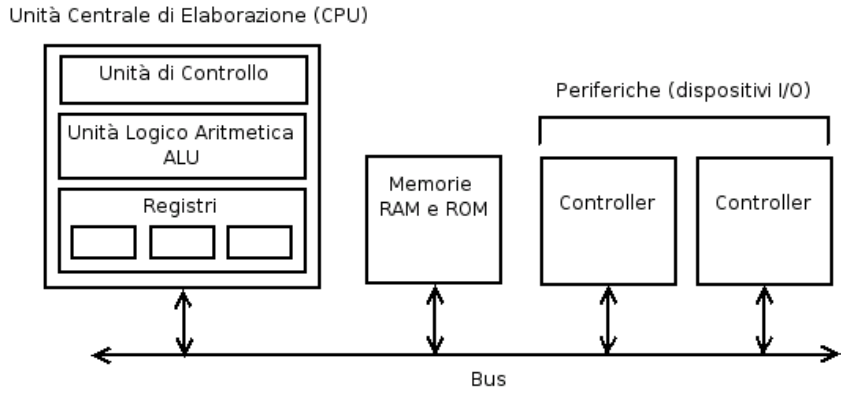
Questi appunti hanno lo scopo di illustrare il funzionamento di un sistema di elaborazione. Vengono trattate le caratteristiche salienti delle parti hardware che compongono l'unità centrale di un PC e come, queste, cooperano affinché il computer possa svolgere le elaborazioni richieste.

Accanto alla descrizione delle varie componenti hardware di un computer, e al modo in cui interagiscono fra di loro, viene considerato il formato delle istruzioni eseguibili da una CPU e il modo come un programma, conservato in formato eseguibile, viene eseguito dalla CPU.

Affinché si possa seguire la trattazione degli argomenti sono necessarie nozioni generali di Informatica e del mondo del computer: bit, byte, hardware e software. Qualche conoscenza basilare su come è fatto un programma in C/C++, può essere di aiuto in qualche esempio riportato.

Struttura a bus

Un computer, dal punto di vista hardware, è composto da un sistema di processori, memorie, controller per i collegamenti alle periferiche. Dispositivi interconnessi fra di loro per poter effettuare le elaborazioni richieste.

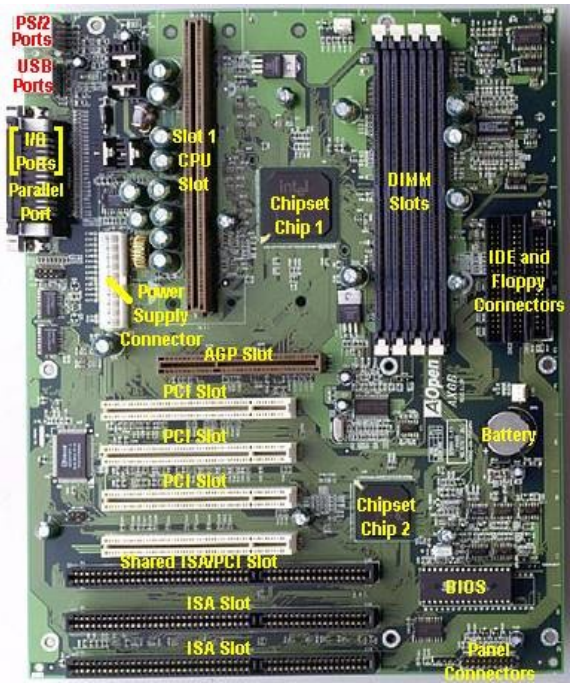


I componenti sono collegati e comunicano utilizzando stringhe binarie che vengono inserite nel bus che, sostanzialmente, si compone, concettualmente, di un insieme di fili paralleli (piste di rame nella realtà) che permettono il passaggio dei segnali elettrici fra un componente e l'altro. Quello mostrato in figura è il bus esterno; esistono anche dei bus interni, per esempio, nella CPU per permettere il passaggio di segnali fra le varie unità che la compongono.

I controller sono dispositivi elettronici che consentono l'uso delle unità di Input/Output. Una unità di I/O, o periferica, è composta da una parte meccanica che è la periferica stessa (la stampante, un HD, un mouse ...) e una parte elettronica, il controller appunto, che governa la parte meccanica.

Un importante vantaggio del modello a bus è quello di essere *modulare*: si può, per esempio aggiungere, in qualsiasi momento, una periferica: basta inserire la scheda-controller e collegare il cavo della periferica. Allo stesso modo, se una periferica non funziona o si vuole sostituire, basta rimuovere il controller senza che questa operazione abbia una qualche influenza sul funzionamento del resto dei dispositivi.

La CPU, nei PC, è contenuta, fisicamente, in un unico circuito che viene chiamato **microprocessore** (spesso abbreviato in μP). Certe volte vengono usati i due termini, processore (microprocessore) e CPU, come sinonimi anche se le funzioni di una CPU potrebbero essere svolte da più processori.



(immagine tratta da http://www.kids-online.net)

Fisicamente il *cuore* di un computer è la **motherboard** (scheda madre): circuito stampato che contiene tutti i circuiti per interfacciarsi con i vari componenti e il bus.

Nell'immagine, riportata a lato, si notano:

- gli slot di espansione (in basso a sinistra): sono i connettori sui quali si inseriscono i controller per le periferiche, ma anche (immediatamente sopra) per inserire una scheda per la CPU. In alcuni casi la CPU è inserita in un apposito alloggiamento saldato nella scheda. Sulla destra, in alto, si possono notare, inoltre, gli slots per inserire le schede di memoria
- vari circuiti di servizio per gestire i collegamenti
- la ROM che contiene il BIOS (in basso a destra) per gestire la fase di avvio

Anche se il bus fisicamente è un insieme unico di collegamenti per lo scambio di informazioni fra i vari componenti ad esso collegati, si possono, dal punto di vista logico, distinguere tre bus, in ragione del tipo di dati che viaggiano in essi:

- **Data Bus** (Bus dati): in questo bus viaggiano i dati che si scambiano i vari componenti. Per esempio i dati che dalla memoria centrale vanno verso la CPU per essere elaborati o quelli che dalla CPU vanno verso la memoria per essere conservati.
- **Address Bus** (Bus indirizzi): in questo bus viaggiano gli indirizzi di memoria cui si vuole accedere. Come si esaminerà più avanti, i posti (le locazioni) in cui si conservano i dati, sono identificati da numeri, gli indirizzi appunto. Ad ogni locazione o registro di memoria accessibile è associato, in modo univoco, un indirizzo. Si fa notare che registri di memoria sono presenti anche nelle interfacce verso le periferiche: i dati che vanno, per esempio, verso una stampante, vengono depositati in tali registri. Sarà poi il controller che si occuperà di gestire la stampa fisica dei dati.
- **Control Bus** (Bus controlli): in questo bus viaggiano tutti i segnali di sincronizzazione fra i vari dispositivi, necessari per la comunicazione.

Se un dispositivo deve comunicare con un altro: invia il comando lungo il bus controlli, l'indirizzo interessato nel bus indirizzi e il dato nel bus dati.

La memoria principale

La memoria principale è costituita da un insieme di dispositivi in grado di conservare, utilizzando stati binari, l'unità di informazione. Negli anni i costruttori di hardware hanno standardizzato la dimensione dell'unità di informazione in 8 bit (1 byte) e, di conseguenza, la memoria è organizzata a byte. Ogni unità in grado di conservare un byte di informazione viene chiamata **cella**, **locazione** o **registro** di memoria. Ad ogni cella di memoria è associato, in maniera biunivoca, un **indirizzo** che

è un numero che identifica la cella e che permette di rintracciarla.

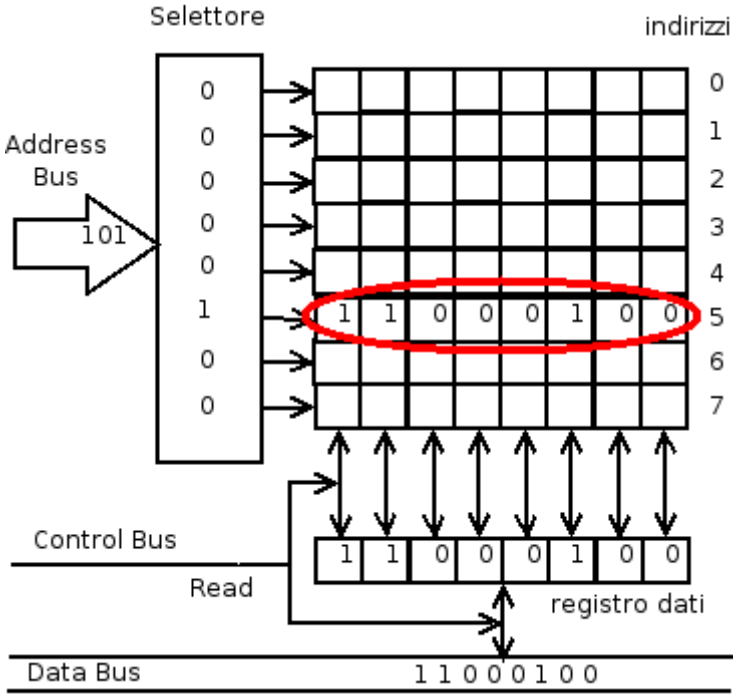
In generale si utilizza il termine registro quando si vuole fare riferimento a dispositivi di memoria che registrano i dati in transito, cella o locazione quando invece si vuole fare riferimento a posti in cui il dato viene conservato in modo da potervi accedere successivamente.

La caratteristica principale della memoria principale, a parte la ROM, è che i dati conservati nelle celle, si distruggono in seguito alla mancanza di alimentazione elettrica (***volatilità*** della memoria). Nella memoria è possibile compiere due operazioni:

- **Accesso in scrittura**: un dato viene copiato in una determinata locazione di memoria. Si tratta di una operazione *distruttiva*: qualsiasi cosa contenuta in precedenza nella cella, viene cancellata e il nuovo dato viene conservato al suo posto.
- **Accesso in lettura**: un dato, conservato in una determinata cella, viene reso disponibile. Si tratta di una operazione *non distruttiva*: si può leggere il dato più volte e lo si ritroverà sempre, finché non interviene una operazione di scrittura che lo distrugge.

La memoria centrale è una ***memoria ad accesso random***. Con tale terminologia si intende il fatto che si può accedere, in qualsiasi momento, a qualsiasi locazione di memoria e, inoltre, la velocità di accesso non dipende dalla storia precedente: è indipendente dalla posizione della locazione di memoria cui si è acceduto in precedenza.

Nel grafico riportato di seguito è esemplificato, per mostrare il funzionamento di una memoria, un accesso in lettura:



nel grafico si suppone, per semplicità, che la memoria sia costituita da 8 locazioni che hanno indirizzi da 0 a 7 e, quindi, in binario, da `000` a `111`.

- Il dispositivo che effettua l'accesso (generalmente la CPU) specifica:
    - nel Control Bus l'ordine di lettura. Il segnale presente in questo Bus influenza il verso dei dati: nel caso in esempio (accesso in lettura), il verso andrà dalla locazione di

memoria al registro e dal registro al Bus Dati

- nell'Address Bus l'indirizzo, della locazione cui si vuole accedere, che andrà nel selettore presente nella memoria. Nell'esempio proposto, l'indirizzo `101`

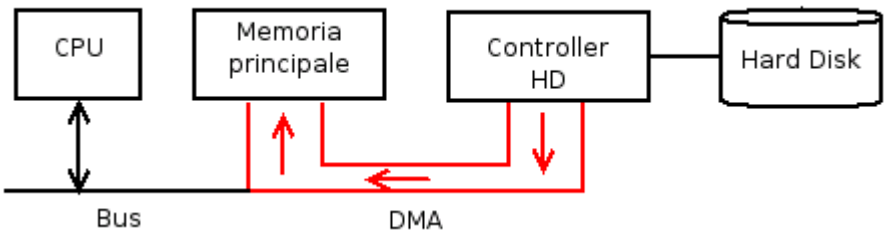
- Nel dispositivo di memoria:
  - ci si predispone all'operazione di lettura
  - il selettore individua la locazione associata all'indirizzo specificato
  - il dato contenuto nella locazione individuata viene ricopiato nel registro dati, presente nella memoria. Questo registro di memoria viene anche chiamato *Memory Buffer*. In Informatica con il termine Buffer si intende, generalmente, una memoria temporanea.
  - il dato viene inviato al Data Bus a disposizione di chi lo ha richiesto

Anche una operazione di scrittura, fatte le dovute differenze, avviene in maniera similare.

Nel caso di un accesso in scrittura dal Control Bus arriva un ordine di scrittura (write), e, in questo caso, il verso dei dati andrà dal Data Bus al registro dati della memoria e dal registro dati della memoria alla locazione selezionata. Dall'Address Bus arriva l'indirizzo della locazione di memoria in cui si vuole salvare il dato che viaggia nel Data Bus. L'indirizzo arriverà, al solito, nel selettore che si occuperà di individuare la locazione di memoria interessata, in modo da ricopiargli il dato che, nel frattempo, è stato depositato nel registro dati.

Collegamenti con le periferiche

Come già accennato, ogni periferica è collegata ad una scheda controller che ha il compito di pilotare la periferica e gestire l'accesso al bus.



Quando la CPU riceve, da un programma, una richiesta di lettura di dati dall'hard disk, invia una richiesta al controller che a sua volta invia all'unità la richiesta per la lettura dei dati. L'unità invia i dati al controller che è in grado di scrivere o leggere in memoria senza fare intervenire la CPU, effettuando quello che viene chiamato **DMA** (Direct Memory Access).

L'approccio di tipo DMA comporta una maggiore velocità dell'elaborazione, specialmente in un contesto di uso massiccio della periferica: l'hard disk è infatti utilizzato non soltanto per conservare dati da elaborare e programmi per la loro elaborazione, ma anche per lo swapping delle pagine dei programmi in esecuzione. L'accesso diretto in memoria consente alla CPU di *svincolarsi* dal trasferimento di dati e, parallelamente al lavoro di trasferimento di dati da parte del controller, effettuare altre elaborazioni.

La CPU, che controlla l'esecuzione di un programma, ha necessità di conoscere i risultati delle operazioni richieste alle periferiche e, tutto ciò, per qualsiasi periferica anche per quelle che non

effettuano accessi diretti in memoria. Si pensi, infatti, ai dati inviati, per esempio, ad una stampante: si ha necessità di sapere se la stampa è andata a buon fine o se, per esempio, non esisteva una stampante collegata, al fine di gestire in maniera opportuna la continuazione del programma.

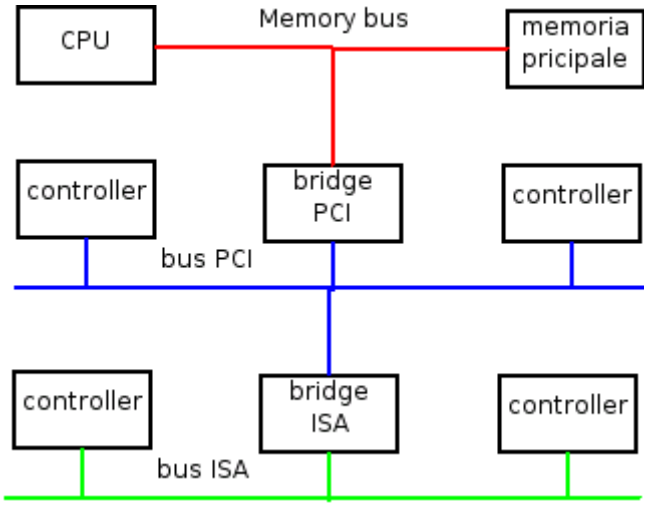
Il problema principale della intercomunicazione fra CPU e periferica è il fatto che gli eventi che deve gestire la CPU sono **eventi asincroni**: non si può prevedere in anticipo quando, per esempio, la stampante finisce il proprio lavoro dipendendo questo da molti fattori variabili come velocità della stampante stessa, quantità di testo, velocità della connessione, distanza ecc...

La gestione degli eventi asincroni viene effettuata con il metodo degli **interrupt** (interruzioni):

1. la periferica quando vuole interloquire con la CPU, lancia un segnale di interrupt per mezzo del controller
2. il segnale viene intercettato da un apposito circuito che notifica alla CPU la presenza di una interruzione da parte di una specifica periferica
3. la CPU sospende il programma in esecuzione in quel momento, e lancia l'**interrupt handler**, programma di servizio dell'interruzione, che gestisce la comunicazione con la periferica che ha richiesto i propri servizi
4. servita l'interruzione, il programma, sospeso in precedenza, viene ripristinato e può, così, continuare la propria esecuzione.

Mano a mano che le CPU e tutti gli altri dispositivi diventavano sempre più veloci, si pose il problema del carico del bus che non riusciva più a gestire il traffico dei dati. Il problema poteva essere risolto progettando un nuovo tipo di bus che permettesse velocità di trasferimento maggiori, ma, in questo caso, gli utenti che avessero voluto utilizzare con i nuovi computer una stampante o un modem acquistati precedentemente, non avrebbero potuto farlo.

La soluzione adottata fu quella dei bus multipli:



Il bus di tipo più vecchio, e lento, è il **bus ISA** (Industry Standard Architecture) sostituito nelle macchine più recenti dal **bus PCI** (Peripheral Component Interconnect) molto più veloce. La CPU dialoga con la memoria principale utilizzando un bus dedicato ad alta velocità.

I collegamenti fra i vari bus sono garantiti da circuiti ponte (bridge PCI e bridge ISA).

Anche nella foto della scheda madre, riportata in una pagina precedente, sono visibili nella parte in

basso a sinistra, gli slot di tipo PCI e di tipo ISA per le connessioni con i relativi controller.

Gli slot di tipo ISA, e quindi il relativo bus, tendono, nelle configurazioni più nuove, a scomparire. Il bus ISA viene ad essere sostituito, per esempio, dal bus PCI-X o dal bus PCI-Express (noto anche con la sigla PCIe o PCIx), quest'ultimo in uso per le schede grafiche.

Caratteristiche della CPU: word, ciclo di clock

La memoria, come si è fatto rilevare, è organizzata a byte e, d'altronde, un singolo carattere di un dato alfanumerico, necessita di un byte per poter essere conservato. Anche per la conservazione in memoria di dati numerici si fa sempre riferimento a multipli del byte.

Nelle CPU più vecchie, l'unità di informazione prelevata o inviata in memoria, in operazioni di lettura o scrittura rispettivamente, era il byte. Con l'aumentare della potenza delle CPU l'unità di informazione prelevata, inviata o elaborata è diventata un multiplo del byte e viene chiamata **word**. Una CPU che adotta una word di 16 bit (***grado di parallelismo***), per ogni accesso in lettura in memoria, legge due byte e le operazioni vengono effettuate considerando 16 bit per volta. Tutti i registri in cui vengono conservati dati, hanno questa dimensione. Nelle CPU attuali vengono utilizzate word di 32 bit (architetture IA-32, Intel Architecture a 32 bit) e si sta migrando verso architetture IA-64 con word di 64 bit.

I compilatori che traducono le istruzioni di un programma, scritto in un linguaggio ad alto livello come il C/C++, in istruzioni eseguibili, mettono a disposizione contenitori della dimensione di una word per rendere più efficiente la gestione della memoria e più veloci le operazioni con dati che vengono conservati in essi. Il dato di tipo `int` del C/C++ è, appunto, un contenitore che ha la dimensione di una word e le operazioni, con dati numerici contenuti in una variabile di questo tipo, vengono svolte in maniera rapida. Non è un caso, per esempio, che nei cicli a contatore (`for`) la variabile che controlla il ciclo, deve essere di quel tipo: la variabile viene modificata ad ogni iterazione e l'operazione è più veloce se la sua dimensione coincide con la dimensione su cui lavora la CPU. Anche i tipi numerici più complessi, `float` o `double`, hanno dimensioni multiple della word.

Anche per gli indirizzi di memoria viene utilizzata una word. Fra l'altro è questo uno dei motivi per cui la dimensione della word è aumentata nel tempo. Una CPU che ha una word di 32 bit può indirizzare fino a 2^{32} = 4.294.967.296 (4 Gb) locazioni di memoria. Questa è la ***potenza di indirizzamento***: il numero massimo di locazioni di memoria accessibili. È la dimensione della word che determina la *limitatezza della memoria principale*: non si possono installare più locazioni di memoria centrale di quante non ne sono indirizzabili perché, semplicemente, non sarebbero accessibili.

Per sincronizzare l'azione della CPU con quella degli altri dispositivi, nella motherboard, è presente un apposito circuito: il **clock** di sistema. La sincronizzazione è importante perché, per esempio in una operazione di scrittura in memoria, il circuito di memoria deve sapere quando è il momento di andare a leggere dal bus, o la CPU deve sapere quando è il momento di prelevare il risultato di una operazione.

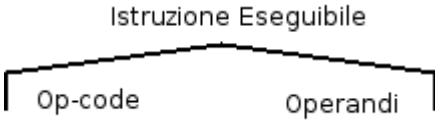
Il clock è un circuito che commuta fra il valore zero e un valore associato ad uno e viceversa. Si definisce ***ciclo di clock*** il tempo trascorso fra due commutazioni di clock. Il ciclo deve svolgersi in

un tempo tale da consentire a tutti i circuiti interessati il completamento corretto dell'operazione effettuata.

Si chiama ***frequenza di clock*** il numero di clock nell'unità di tempo. Maggiore sarà la frequenza e maggiori sono le operazioni che si possono fare nello stesso tempo. La frequenza viene misurata in Hertz: numero di cicli al secondo. Le moderne CPU sono molto veloci e, per misurare la frequenza, vengono utilizzati multipli dell'unità di misura, così si parla di Mhz (megahertz ovvero milioni di cicli al secondo) o Ghz (gigahertz ovvero miliardi di operazioni al secondo).

Istruzioni eseguibili

Un programma scritto in un qualsiasi linguaggio, per esempio C/C++, per poter essere eseguibile da una CPU necessita di un processo di trasformazione (compilazione) effettuato da un compilatore. Il compilatore conosce le istruzioni del linguaggio ma conosce anche le istruzioni che la CPU, per la quale sta compilando, è in grado di eseguire e, quindi, conosce come far corrispondere le une alle altre.



Una istruzione eseguibile da una CPU è una stringa binaria composta da:

- un codice operativo (**op-code**) in cui viene specificata l'operazione da svolgere
- zero o più **operandi** (in dipendenza dell'op-code) che completano l'istruzione e specificano l'obiettivo dell'operazione.

Il codice operativo è scelto dall'**instruction set** del processore che rappresenta l'insieme dei comandi implementati, dal costruttore, in quel processore. L'insieme degli op-code costituisce quello che viene chiamato *linguaggio macchina* di quel processore. La dimensione in bit dell'opcode dipende dalla word su cui lavora il processore, anche se con l'evolversi dell'hardware la dimensione della word è cambiata notevolmente: per esempio nei processori della famiglia x86 (processori inizialmente sviluppati da Intel, ora anche da altri produttori come AMD; la x iniziale sta ad indicare la prima parte degli identificativi della famiglia di processori: 8086, 80386, ...) il primo processore della famiglia, l'8086, prevedeva una word di 16 bit, i moderni processori della stessa famiglia vanno verso word di 64 bit. A questo punto si pongono problemi di retrocompatibilità: affinché i programmi scritti per un processore precedente possano funzionare con le nuove versioni, è necessario che la nuova word debba poter coesistere con dimensioni alternative. L'architettura dei moderni computer è basata su una famiglia di dimensioni di word collegate fra loro (32 è il doppio di 16 e 64 è il quadruplo di 16). Pertanto oggi il termine word viene considerato sinonimo di 16 bit. I codici operativi di un determinato set di istruzioni specificano operazioni molto semplici del tipo: somma, sottrai, sposta, salta all'istruzione, sposta in memoria.

Gli operandi specificano i dati cui si applica l'operazione espressa dall'op-code. Il modo con cui si accede ai dati viene chiamato **metodo di indirizzamento**. In sintesi esistono due tipologie di metodi di indirizzamento:

- **Metodi diretti**. Viene specificata la posizione del dato sia esso in memoria centrale e, in questo caso, ne viene esplicitato l'indirizzo, sia esso in uno dei registri della CPU individuato da una sigla identificativa. L'accesso al dato, in questi casi, è veloce ma si obbliga il dato ad essere

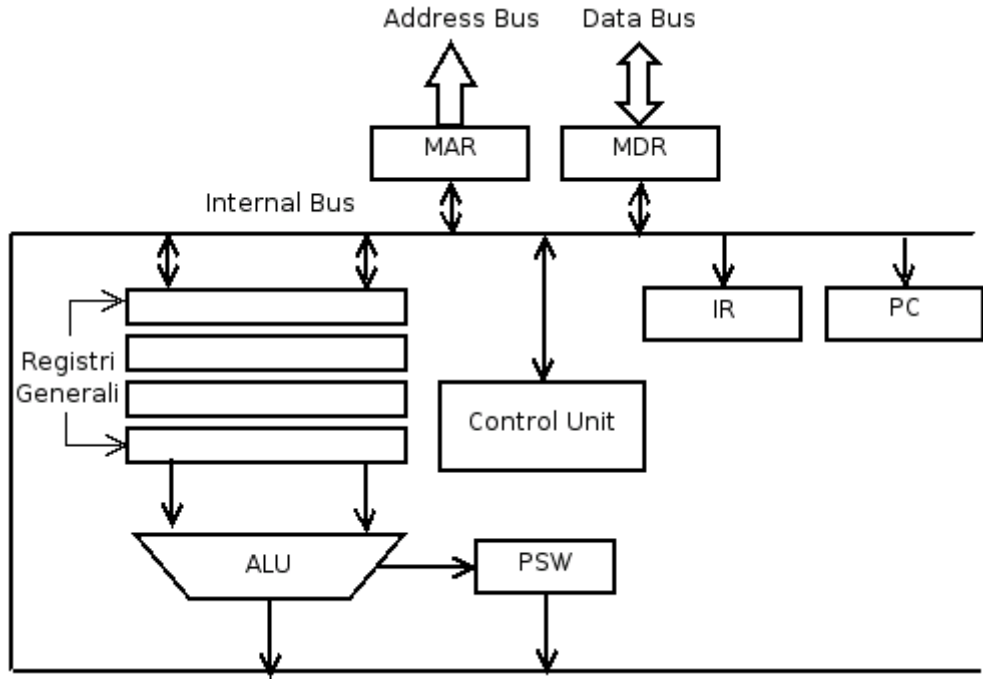
sempre, ad ogni esecuzione del programma, residente nelle stesse posizioni. Questi metodi vengono utilizzati solo in alcuni casi specifici.

- **Metodi indiretti**. La posizione dei dati viene specificata come somma fra uno spiazzamento e un numero che identifica la posizione a partire dalla quale cominciano i conteggi, e che può essere variata. In questo modo, per esempio, si può inserire il programma in una zona qualsiasi di memoria, indicare l'indirizzo di inizio e il programma, sommando a questo lo spiazzamento, saprà dove trovare i dati da elaborare.

Il programma da eseguire, in linguaggio macchina, viene conservato, allo stesso modo dei dati, in una zona di memoria disponibile e può essere avviato.

CPU ed esecuzione di un programma

Per l'esecuzione di un programma in linguaggio macchina, una CPU esegue, iterativamente, un **ciclo di fetch-decode-execute**: preleva (fetch) una istruzione dalla memoria, la decodifica (decode) e la esegue (execute).



Nello schema sono riportati i componenti principali di una CPU:

- L'interfaccia con i bus: il Memory Address Register e il Memory Data Register che permettono alla CPU di dialogare con le altre componenti.
- Un bus interno che consente lo scambio di dati fra i vari componenti interni alla CPU e con i bus esterni (Internal Bus).
- I registri generali utilizzati per depositare temporaneamente i dati su cui si stanno effettuando le elaborazioni.
- L'ALU (Arithmetic Logic Unit) che effettua le operazioni aritmetiche e logiche.
- L'Unità di Controllo che ha il compito di coordinare le altre componenti presenti nel processore. Per esempio abilita alla scrittura o alla lettura i vari registri, interpreta le istruzioni. Ad ogni ciclo

di clock invia determinati segnali, in dipendenza dell'interpretazione dell'istruzione, utili per l'esecuzione di una istruzione. I segnali sono ricevuti dalle componenti interessate che possono, così, predisporsi in modo corretto al compito da svolgere.

- Il registro PSW (Program Status Word) in cui viene registrato lo stato delle operazioni effettuate.
- I registri IR (Instruction Register) e PC (Program Counter) per puntare all'istruzione corrente o alla prossima istruzione da eseguire.

Il ciclo fetch-decode-execute eseguito dal processore per ogni istruzione, si può esprimere, tenendo conto del ruolo dei registri:

Fase di fetch	1. Per avviare un programma, residente nella memoria principale, bisogna inserire l'indirizzo della prima istruzione nel Program Counter. 2. La CU ricopia il contenuto del PC nel MAR e viene effettuato un accesso in lettura a quell'indirizzo. 3. Il dato prelevato dall'indirizzo, cui si è acceduto, viene conservato in MDR. 4. Il contenuto di MDR viene ricopiato in IR. Nel frattempo PC si è, in maniera automatica, aggiornato alla locazione successiva.
Fase di decode	L'istruzione viene interpretata, prelevando eventualmente dalla memoria altri dati necessari se l'istruzione completa non è contenuta nel dato prelevato in precedenza. In ogni caso, alla fine, PC punterà all'istruzione da eseguire successivamente.
Fase di execute	Se, per esempio, l'istruzione da eseguire è una somma, i registri verranno caricati con i numeri da sommare e la CU invia alla ALU il comando per l'esecuzione della somma. La ALU esegue l'operazione: il risultato prodotto sarà conservato in uno dei registri e verrà aggiornata la PSW.

Il ciclo viene ripetuto per la prossima istruzione fino a che non si arrivi ad una istruzione di halt.

La PSW o **registro di stato** conserva lo stato di alcune operazioni matematiche ed è un registro che funziona in modo particolare: ogni singolo bit ha un significato autonomo a differenza degli altri registri dove ha significato l'intera stringa binaria. Ogni singolo bit del registro di stato è un indicatore (**flag**) di un determinato evento che si è verificato nel corso dell'ultima operazione.

Di seguito si esaminano alcuni indicativi:

- **CF**: Carry Flag o flag del riporto. Indica se l'operazione effettuata ha prodotto un risultato non contenibile nello spazio predisposto. Per esempio se si effettua una operazione a 16 bit, indica il riporto al 17-esimo bit.
- **OF**: Overflow Flag. Indica se nell'operazione effettuata si è verificato un errore di overflow (risultato non rappresentabile nello spazio predisposto).
- **IF**: Interrupt Flag. Abilita o disabilita le interruzioni. Ogni volta che è terminato un ciclo di fetch-decode-execute il processore può essere interrotto da una periferica; interruzione che non si può accettare durante le tre fasi che comportano l'esecuzione di una istruzione e che devono essere eseguite, anche se prevedono diversi accessi in memoria, come se fosse un unica operazione. Un interrupt da parte di una periferica comporta, se accettato, che la CU conservi lo

stato del programma attualmente in esecuzione (il contenuto del PC e dei registri generali) in una apposita area di memoria (lo **Stack di sistema**), carichi nel PC l'indirizzo del programma di servizio dell'interruzione e, alla fine di questo, ripristini dallo stack i registri, in modo da continuare l'esecuzione del programma sospeso.

- **ZF**: Zero Flag. Indica se il risultato dell'operazione effettuata è stato nullo.
- **SF**: Sign Flag. Praticamente uguale al bit più significativo del risultato di una operazione, ne indica il segno.

Normalmente le istruzioni di un programma vengono eseguite in sequenza, ma, in presenza di strutture di controllo, la sequenza potrebbe essere modificata: si pensi, per esempio in C/C++, a strutture come `if` o `for`.

Gli ultimi due flag vengono utilizzati quando si tratta di effettuare salti condizionati.

```
...
// istruzioni A
if(alfa>beta)
  // istruzioni B
else
  // istruzioni C
// istruzioni D
```

Nel frammento di codice riportato, dopo l'esecuzione delle istruzioni A c'è una condizione che specifica, in dipendenza del risultato del confronto, quali istruzioni eseguire prime delle D: se la condizione è vera si eseguono le B e si salta alle D, se è falsa si salta alle C e quindi, in sequenza, si eseguono le D.

Per poter stabilire l'obiettivo del salto è necessario verificare il *risultato* del confronto. Il confronto viene risolto effettuando una *sottrazione senza conservazione del risultato* fra `alfa` e `beta` e consultando i flag di Zero e di Segno.

Considerando che ZF viene posto ad 1 se il risultato è nullo e SF viene posto ad uno se il risultato è negativo, esaminando il valore dei due flag, si possono avere indicazioni sul risultato del confronto:

Operazione	*SF*	*ZF*	*Risultato*
`alfa - beta`	0	0	`alfa > beta`
	0	1	`alfa == beta`
	1	0	`alfa < beta`

Evoluzione dei processori

Le prime CPU avevano una architettura semplice. Mano a mano che le esigenze elaborative aumentavano, si richiedevano esigenze di calcolo superiore e si cercò, e si cerca tuttora, di studiare il modo come aumentare la potenza dei nuovi processori.

In linea generale si può dire che l'evoluzione dei processori si è sviluppata su tre direttive:

- **RISC** e **CISC**. Il tentativo di realizzare CPU sempre più potenti ha portato, inizialmente, a implementare istruzioni sempre più complesse che avvicinassero il linguaggio macchina a un linguaggio ad alto livello. Per esempio per tradurre più semplicemente una istruzione del tipo `a=b+c`, si mise a disposizione un op-code che era in grado, in un unica istruzione, di effettuare un operazione simile, fornendo gli indirizzi delle locazioni di memoria corrispondenti alle variabili

specificate. Questa operazione era poi, dall'hardware, scomposta in altre più elementari che portavano nei registri i dati, effettuavano la somma e riportavano in memoria il risultato. Le istruzioni diventavano più complesse e si allungavano i cicli di clock necessari per eseguire l'istruzione. Un gruppo di ricercatori propose un approccio diverso: dotare i processori di istruzioni semplici che potessero essere eseguite in un unico ciclo di clock. Il programma sarebbe stato più lungo ma sarebbe stato eseguito più velocemente: nacquero i processori ad architettura RISC (Reduced Instruction Set Computer) in contrapposizione all'altro tipo di architettura che prese il nome CISC (Complex Instruction Set Computer).

L'esecuzione di una operazione di somma, in linguaggio macchina, nei due tipi di architettura, usando un linguaggio simbolico, supponendo che alla variabile `a` corrisponda l'indirizzo di memoria 1000 e che alla variabile `b` corrisponda 1001, si potrebbe esprimere:

Operazione	*CISC*	*RISC*

a = a + b	`Somma 1000 1001`	`1. Carica nel registro A il valore contenuto in 1000`
		`2. Carica nel registro B il valore contenuto in 1001`
		`3. Somma reg.A e reg.B e salva in reg.A`
		`4. Copia in 1000 il contenuto del registro A`

L'approccio di tipo CISC tende a demandare all'hardware la risoluzione delle operazioni più semplici, include operazioni complesse multi-clock, tende a far risparmiare memoria.

L'approccio di tipo RISC sottolinea l'aspetto software, include operazioni semplici da svolgere in un unico ciclo di clock, produce codice più lungo.

Per un po' di tempo ci fu una contrapposizione netta fra i due approcci finché Intel pose fine alla disputa utilizzando, dal 486, un approccio diverso: le CPU contengono un piccolo processore RISC per l'esecuzione delle istruzioni più comuni e semplici; quelle meno comuni vengono interpretate nel modo CISC.

- **Pipelining**. Un modo per accelerare il funzionamento dei processori è quello di aumentare la velocità del clock, ma c'è un limite tecnologico oltre al quale non si può andare. Pertanto i ricercatori si rivolsero alla ricerca di altre soluzioni. La soluzione adottata consiste nel parallelismo: fare eseguire dal processore più operazioni contemporaneamente. Si tratta, in breve, di scindere il ciclo fetch-decode-execute in piccole parti sovrapponibili: quando una parte esegue il primo stadio di una istruzione e passa al secondo, ci sarà un'altra parte che comincia l'esecuzione del primo stadio dell'istruzione successiva. In questo modo se, per esempio, si suddivide il ciclo in 3 stadi (fetch, decode, execute), a partire dal terzo stadio si avrà il completamento di una istruzione per ogni stadio.

  Le CPU moderne utilizzano molti più stadi dei 3 dell'esempio precedente: il Pentium 4 ne utilizza da 20 a 30.

- **Multi Core**. Un altro modo utilizzato per aumentare le prestazioni di un processore, quando l'aumento della velocità del clock diventa complicato da realizzare, è quello di portare il parallelismo anche a livello hardware: due CPU indipendenti vengono assemblate assieme in un unico pacchetto in modo da lavorare assieme, eseguendo istruzioni diverse, all'esecuzione dello stesso programma. Si tratta dei cosiddetti processori **dual core** cioè processori con due nuclei (due *core*). Oltre ai dual core, stanno cominciando a fare sentire la loro presenza anche processori **quad core**: due dual core che lavorano assieme.

Riferimenti

Per la scrittura di questi appunti sono state consultate le seguenti fonti:

- *Architettura dei computer* di Andrew S. Tanenbaum. Testo utilizzando universalmente e punto di riferimento assoluto sul mondo del computer.
- *http://www.wikipedia.org* anche nella edizione italiana *it.wikipedia.org*. L'enciclopedia libera in rete. Risorsa globale diventata ormai un punto di riferimento della rete.