

Terzo capitolo

Livello logico digitale

Algebra di Boole

L'algebra di Boole è una particolare tipologia di algebra in cui le variabili e le funzioni possono avere valori 0 e 1.

Si studia l'algebra booleana poiché le funzioni dell'algebra booleana sono isomorfe ai circuiti digitali. In altre parole, un **circuito digitale può essere espresso tramite un'espressione booleana e viceversa**.

Una funzione booleana ha una o più variabili in input e fornisce risultati che dipendono solo da queste variabili.

Poiché le variabili possono assumere solo i valori 0 o 1 una funzione booleana con n variabili di input ha solo 2^n combinazioni possibili e può essere descritta dando una tabella, detta tabella di verità, con 2^n righe.

Proprietà dell'algebra di Boole

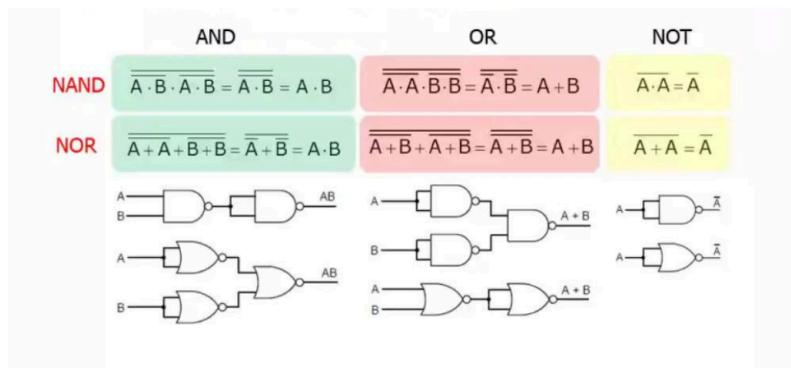
| and form | or form |
|--------------------------------------------------|--------------------------------------------------|
| $1A = A$ | $0 + A = A$ |
| $0A = 0$ | $1 + A = 1$ |
| $AA = A$ | $A + A = A$ |
| $AB = BA$ | $A + B = B + A$ |
| $(AB)C = A(BC)$ | $(A + B) + C = A + (B + C)$ |
| $A + BC = (A+B)(A+C)$ | $A(B+C) = AB + AC$ |
| $A(A + B) = A$ | $A + AB = A$ |
| $\text{not}(AB) = \text{not}(A) + \text{not}(B)$ | $\text{not}(A+B) = \text{not}(A) \text{ not}(B)$ |

Esiste la seguente equivalenza:

- Per ogni espressione logica → esiste un circuito digitale equivalente → una colonna della tavola di verità
- Per ogni colonna della tavola di verità → esiste un'espressione che la rappresenta → esiste un corrispondente circuito digitale
- Per ogni circuito digitale → esiste un'espressione che lo descrive → esiste una corrispondente colonna della tavola di verità

Gli operatori universali

Sappiamo che ogni funzione logica booleana si può realizzare con gli operatori AND, OR e NOT. È possibile dimostrare che questi tre operatori possano essere realizzati con una sola porta NAND o NOR.



Circuiti integrati

Le porte logiche sono vendute individualmente ma dentro i **circuiti integrati (o chip)**. Sono dei pezzi di silicio sul quale sono stampati dei circuiti. Possono essere divisi in classi in funzione del numero di porte logiche che contengono:

- **SSI** → < 10 porte
- **MSI** → 10 < porte < 100
- **LSI** → 100 < porte < 100000
- **VLSI** → > 100000 porte

Circuiti combinatori e sequenziali

- **Circuiti combinatori** → circuiti digitali nei quali l'uscita dipende esclusivamente dagli ingressi, e non dallo stato del circuito, come avviene nei latch;
- **Circuiti sequenziali** → l'uscita dipende sia dagli ingressi che dallo stato del circuito.

Multiplexer

Un **mux** prende in input 2^n bit + n bit detti **di selezione**, e manda in output un valore y tale che

$$y = x_{(s_n-1, \dots, s_0)}_2$$

In altre parole, il circuito dà in output il bit contenuto nei dati di input, ricavato dalla codifica in binario delle variabili di selezione, ottenendo l'indice.

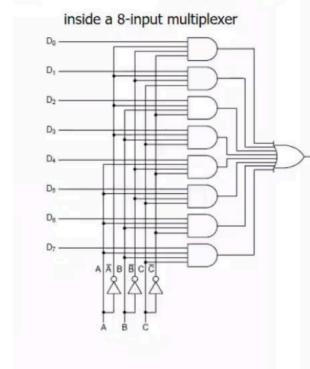
Esempio:

$$\text{Input: } (x_3, x_2, x_1, x_0) = (0, 1, 1, 0)$$

$$(s_1, s_0) = (1, 0)$$

$$(s_1 s_0) = (10)_2 = 2 = i \rightarrow 2 \text{ è l'indice}$$

Quindi $x_2 = 1 = y \rightarrow \text{Output}$



Decoder

Un **decoder** prende in input n bit e manda in output 2^n bit tali che

- $y = 1$ se $(i)_10 = (x(n-1) \dots x_0)_2$
- $y = 0$ altrimenti

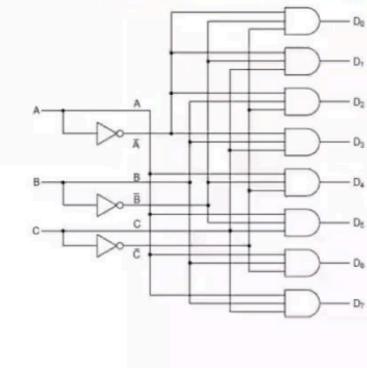
In altre parole, il circuito attiva una delle sue uscite a seconda della combinazione di valori in input. Decodifica la sequenza di bit in input ponendo a 1 l'output y_i , dove i è il numero rappresentato in binario dagli n bit in input.

Esempio:

$$\text{Input: } (x_1, x_2) = (1, 0) = 2 = i$$

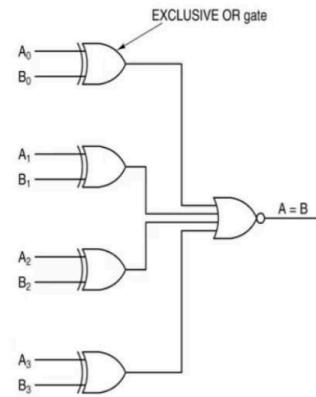
$$\text{Output: } (0, 1, 0, 0) \text{ poiché } (2)_10 = (0010)_2 = (x_1 \times 0)$$

Schema digitale di un decoder a 3 ingressi



Comparatore

Il circuito compara due parole A e B di 4 bit. Il risultato è 1 se i bit della prima parola sono uguali a quelli della seconda, ovvero, per ogni $i = 0, 1, 2, 3$, $A_i = B_i$.



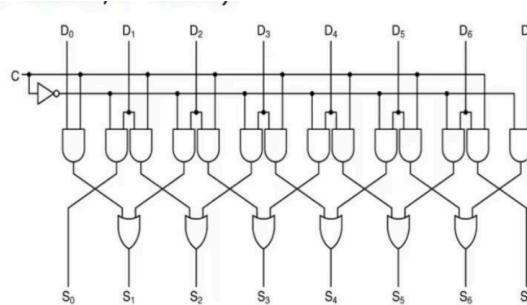
PLA - Array Logico Programmabile

Questa tipologia di circuito permette di calcolare **somme di prodotti**.

Questo chip ha 12 ingressi e, ogni valore in input, ha anche il suo equivalente invertito, per un totale di 24 segnali di input.

Il cuore del circuito è una 50ina di porte AND che possono avere un sottoinsieme dei 24 segnali di input. Poiché sono solo 50, e avremmo all'incirca 2^{24} combinazioni, la scelta del sottoinsieme dipende dal programmatore: ogni linea di input connessa alle porte AND contiene un fusibile, i quali verranno poi bruciati dal programmatore con l'applicazione di un'alta tensione.

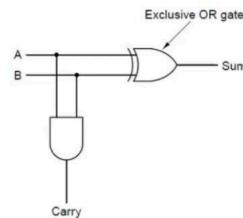
Gli output delle porte AND diventeranno input per le 6 porte OR, ognuna delle quali avrà in ingresso 50 valori in input, sempre programmabili dall'utente.



Half Adder - Semi Sommatore

Dati due bit a e b, questo circuito calcolerà **S** (somma tra i due bit tramite XOR) e **Cout** (bit di riporto tramite AND)

| A | B | Sum | Carry |
|---|---|-----|-------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



31

Il problema fondamentale di questo circuito è il seguente: non tiene conto del riporto di somme precedenti:

Nel caso in cui dovessimo calcolare la somma tra 0001 e 0101, avremmo che il primo valore avrà $S = 0$ e $Cout = 1$; nel calcolo del secondo bit non tiene conto del bit di riporto calcolato in precedenza.

Full Adder - Sommatore

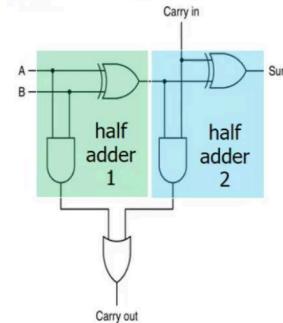
Con il full adder, risolviamo la problematica descritta in precedenza. Infatti questo circuito prende in input i due bit a e b e il bit Cin.

Avremmo che:

$$S = (a \text{ XOR } b) \text{ XOR } \text{Cin}$$

$$\text{Cout} = ab + (a \text{ XOR } b) \text{ Cin}$$

| A | B | Carry in | Sum | Carry out |
|---|---|----------|-----|-----------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

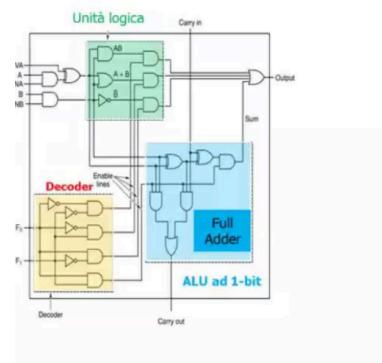


Mettendo in sequenza i full adder otteniamo il circuito **sommatore**.

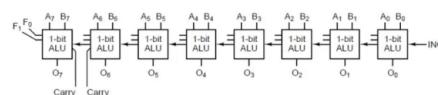
ALU

La ALU è composta da tre unità: un **decoder**, una **unità logica** e una **full adder**:

- **decodificatore** → genera i segnali di attivazione delle 4 operazioni;
In base ai segnali di controllo F0 e F1, viene selezionata una delle quattro linee di attivazione, permettendo all'output della funzione selezionata di passare attraverso la porta logica OR che genera l'output finale;
- **unità logica** → in base alle linee di attivazione, un solo risultato passa nella porta OR; Calcola AB, A + B e not(B);
- **sommatore** → calcola la somma di A e B e gestisce i riporti.



Questi circuiti sono attualmente disponibili e vengono chiamati **bit slices** ("suddivisioni di un bit")

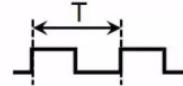


Un ALU a 8-bit costruita con otto ALU ad un 1-bit.

Clock

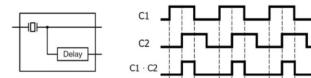
Un **clock** è un circuito che emette una serie di impulsi di larghezza definita e a intervalli temporali costanti. Il clock viene utilizzato principalmente per la gestione della sincronizzazione e permette ai progettisti di ottenere le relazioni temporali desiderate.

Un **ciclo di clock** è l'intervallo temporale compreso tra le estremità di due impulsi consecutivi



In un calcolatore possono verificarsi più eventi durante uno stesso ciclo di clock. Se però è necessario che si verifichino in uno specifico ordine, occorre dividere il ciclo di clock in sottocicli. Una tecnica utilizzata consiste nell'intercettare la linea del clock principale e inserirla in un circuito di cui si conosce il ritardo. In questo modo è possibile generare un secondo segnale di clock la cui fase è traslata rispetto a quella del clock principale.

Una tecnica che permette di aumentare la risoluzione del segnale di clock C1 è di effettuare un AND tra il segnale originario e una sua replica ritardata C2.



La memoria

La **memoria** è utilizzata per conservare sia le istruzioni da eseguire sia i dati

I circuiti combinatori non sono adatti per la costruzione di un circuito che memorizza i dati, poiché **non hanno un percorso**: l'output non diventa l'input della stessa porta. Una **rete combinatoria** è la soluzione più adeguata per realizzare una memoria.

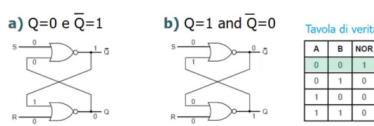
Un **circuito sequenziale** è un circuito in cui l'output dipende sia dagli **ingressi** che dallo **stato del circuito**.

SR - Latch

SR Latch è un circuito sequenziale composto da due porte NOR. Ogni porta NOR prende in input un bit S (**set**) o R (**reset**) e il valore in output dell'altra porta NOR.

Il bit **set S** serve per impostare il valore del latch; Il bit **reset R** per azzerarlo.

Gli output Q e not(Q) sono uno il complementare dell'altro



| S | R | Q |
|---|---|---|
| 0 | 0 | ? |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Porta NOR

- Se uno dei due elementi = 1 \rightarrow output = 0
- Se entrambi = 0 \rightarrow output = 1

Nel caso in cui $(S,R) = (0,0)$:

Non sappiamo le uscite in output. Nel caso in cui:

- not(Q) = 0** \rightarrow In output con R avremmo Q = 1; Q con S = 0, avremmo in output not(Q)=0;
- not(Q) = 1** \rightarrow In output con R avremmo Q = 0; Q con S = 0, avremmo in output not(Q)=1.

| S | R | Q | not(Q) |
|---|---|---------|--------------|
| 0 | 0 | Q(prec) | not(Q)(prec) |
| 0 | 1 | 1 | 0 |

Quindi, gli output di Q e not(Q) dipenderanno da Q(prec) e not(Q)(prec).

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

Due problematiche:

- L'ultimo stato (R,S) = (1,1) fa sì che (Q,not(Q)) = (0,0), quindi non posso dire che not(Q) è complementare a Q;
- Dati (R,S) = (1,1), cosa succede se poi passiamo direttamente a (R,S) = (0,0)? \rightarrow Il circuito non è più stabile.

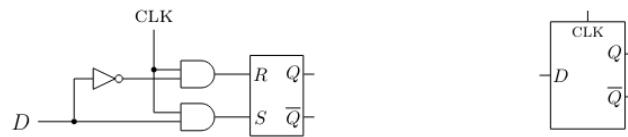
D - Latch

Il **D-Latch** possiede due ingressi:

- Un **ingresso dati D**, il quale controlla **il prossimo stato**;
- Un **ingresso clock CLK**, il quale controlla **il cambio dello stato**.

L'obiettivo di questo circuito è:

- Garantire che Q e not(Q) siano complementari;
- Consente di separare **quale** bit viene assegnato a Q **da quando** viene assegnato.



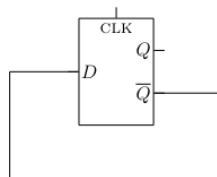
| CLK | D | not(D) | R | S | Q | not(Q) |
|-----|---|--------|---|---|---------|--------------|
| 0 | 0 | 1 | 0 | 0 | Q(prec) | not(Q)(prec) |
| 0 | 1 | 0 | 0 | 0 | Q(prec) | not(Q)(prec) |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |

Quando

- CLK = 1 \rightarrow Latch è **trasparente**;
- CLK = 0 \rightarrow Latch è **opaco** \rightarrow viene bloccato il passaggio dei dati verso Q, che mantiene il valore precedente

Flip - Flop

Il fatto che il latch è trasparente quando CLK = 1 può essere uno svantaggio. Supponiamo di avere il seguente circuito:



- Se CLK = 0 \rightarrow not(Q) mantiene il valore precedente, quindi non è un problema;
- Se CLK = 1 \rightarrow **paradosso logico**.

La problematica si può risolvere mettendo in sequenza due D-Latch collegati da un CLK:

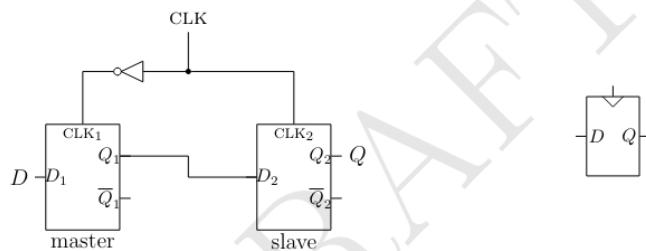


Figura 3: Schema e simbolo di un D-FlipFlop

- Se $\text{CLK} = 0$
 - **master trasparente** (entra con $\text{CLK} = 1$)
 - **slave opaco**

Quindi Q_1 diventa input di D_2
- Se $\text{CLK} = 1$
 - **master opaco**
 - **slave trasparente**

Quindi D_2 passa a Q

C'è un passaggio $D \rightarrow Q$ se e solo se $\text{CLK} = 0 \rightarrow \text{CLK} = 1$

Una serie di Flip-Flop con i clock sincronizzati costituiscono un **registro**:

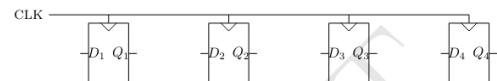
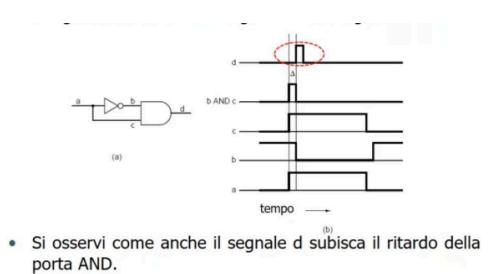


Figura 4: Un registro a quattro bit

Generatore di impulsi

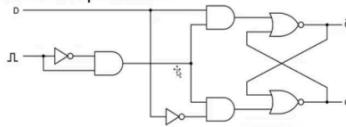
Un generatore di impulsi si può realizzare utilizzando una porta logica AND e un inverter che ritarda il segnale in uno dei due ingressi come in figura.



- Si osservi come anche il segnale d subisca il ritardo della porta AND.

Questo circuito può sostituire il CLK nel D-Latch e permette di campionare il valore dell'ingresso D in un intervallo di tempo ridotto, formadosi così un'altra versione del Flip-Flop

- Il generatore di impulsi sostituisce il segnale di clock e permette di campionare il valore dell'ingresso D in un intervallo di tempo ridotto.

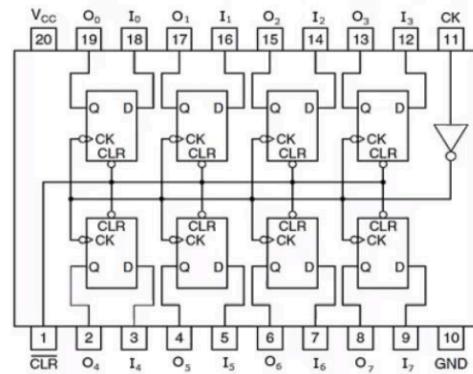


- Molti latch hanno due ingressi aggiuntivi Preset (per forzare lo stato $Q=1$) e Clear (per forzare lo stato $Q=0$).

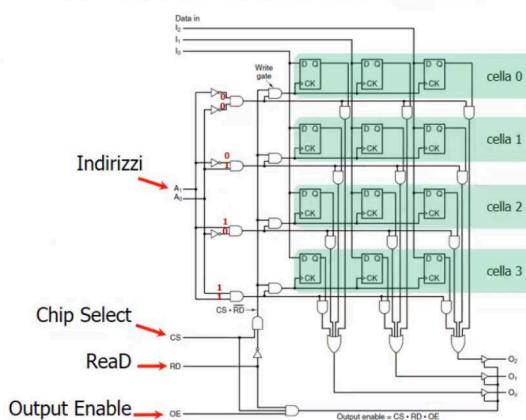
I segnali clear (\overline{CLR}) e preset (\overline{PR}) spesso sono in logica negativa: sono attivi quando sono al livello logico basso.

Registro

Per realizzare un registro a 1 byte necessitiamo di 8 flip-flop D come mostrato in figura. Il registro riceve in ingresso un valore di 8 bit quando vi è una transizione del clock CK. Per implementare un registro, tutte le linee di clock sono collegate allo stesso segnale in ingresso CK, in modo che quando si ha una transizione di clock il registro riceve dal canale di ingresso il nuovo dato di 8 bit.



Organizzazione della memoria



Per realizzare memorie di dimensioni maggiori è necessaria un'organizzazione differente, nella quale sia possibile indirizzare singole parole.

L'immagine mostra una memoria con quattro parole a 3 bit cui ciascuna operazione legge o scrive un'intera parola.

Tale circuito ha 8 bit di input, di cui 3 sono segnali di controllo:

- $I_0 - I_2$ per i dati;
- $A_0 - A_1$ per indirizzare le celle;
- CS per selezionare il circuito;

- RD per selezionare il tipo di operazione;
 - OE per abilitare le uscite (in caso di lettura);
- E 3 bit per l'uscita ($O_0 - O_2$).

Per utilizzare questo circuito è necessario che una componente logica esterna imposta:

- CS * not(RD) = 0 → in caso di lettura
- CS * not(RD) = 1 → in caso di scrittura

L'immagine in alto a sinistra non è altro che un decodificatore: in base ai valori A_0 e A_1 , abilita una delle quattro celle. Le porte AND non sono altro che porte di scrittura, che prende in input l'output di CS * not(RD) e l'output del decodificatore: in questo modo, in caso di scrittura, tre delle 4 porte AND avranno valore 0, mentre una avrà valore 1, abilitando la cella per la modifica. L'output della porta di scrittura guida tutti i segnali CK relativi alla parola selezionata, caricando i dati di input nei flip-flop che costituiscono la parola stessa.

Per l'operazione di lettura, avviene nel seguente modo: quando CS * not(RD) = 0, vengono disabilitate le porte di scrittura (gli AND avranno in output tutti valore 0), impedendo la modifica dei flip-flop. La linea per la selezione della parola scelta abilita le porte AND cui sono collegati i bit Q della parola selezionata. Tale parola spedisce i propri dati alle porte OR in basso. Poiché le altre parole generano in output valori 0, il risultato delle porte OR è identico al valore memorizzato nella parola selezionata.

Chip di memoria

Fissata una dimensione della memoria, esistono diversi modi per organizzare il chip:

Primo caso

Nel primo caso, sono necessarie 19 linee $A_0 - A_{18}$ per indirizzare uno dei 524288 byte e 8 linee di output per caricare e memorizzare i byte selezionati.

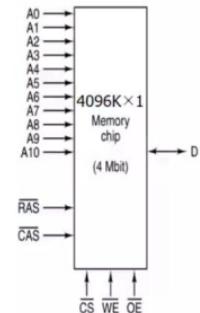
Generalmente, poiché un calcolatore ha di solito vari chip di memoria, sono necessari alcuni segnali fondamentali:

- CS → permette di selezionare il chip da attivare;
- WE → determina la tipologia di operazione;
- OE → consente di fondere insieme i segnali di uscita.

Secondo caso

Nel secondo caso, invece, si ha una matrice 2048×2048 celle a 1 bit. Per indirizzare il chip si seleziona inizialmente una riga immettendo un numero a 11 bit sui pin dell'indirizzo e asserendo il segnale RAS (Row Address Strobe). Dopodiché si immette sui pin dell'indirizzo un numero di colonna e si asserisce il segnale CAS (Column Address Strobe).

Generalmente, i chip di memoria di grandi dimensioni sono spesso composti come matrici n x n. In questo modo, si riduce il numero di pin da utilizzare, rendendo però lento il chip, in quanto sono necessari due cicli di indirizzamento



RAM e ROM

Le RAM (Random Access Memories) sono memorie che possono essere sia lette che scritte.

Esistono due tipologie di RAM:

- **RAM statiche (SRAM)**
 - costruite usando circuiti simili ai flip-flop D;
 - proprietà di mantenere il proprio contenuto fin quando c'è alimentazione;
 - molto veloci e tempi di accesso dell'ordine dei nanosecondi;
 - usate principalmente come memorie cache di secondo livello.
- **RAM dinamiche (DRAM)**

- composte da un array di celle, ciascuna delle quali contiene un transistor e un piccolo condensatore che può essere caricato (1) o scaricato (0). Poiché la carica elettrica tende a disperdersi occorre effettuare una ricarica di ciascun bit della RAM, per evitare che i dati vengano persi;
- hanno una maggiore capacità rispetto alle SRAM, ma richiedono circuiti più complessi per l'interfacciamento.

Esiste anche una versione ibrida, detta **SDRAM**: è guidata dal clock principale del sistema, dando il vantaggio di eliminare la necessità dei segnali di controllo per specificare al chip quando deve rispondere.

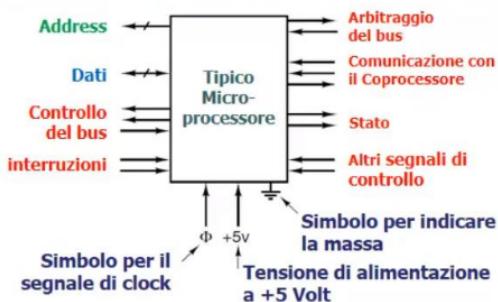
Le ROM (Read-Only Memory) sono memorie nelle quali i dati rimangono memorizzati e non possono essere modificati.

Tipologie di ROM:

- **PROM - ROM programmabili**
 - può essere programmata una volta, bruciando dei fusibili collocati nelle intersezioni della matrice.
- **EPROM - PROM cancellabili**
 - può essere riprogrammata attraverso l'esposizione ai raggi ultravioletti per 15 minuti;
- **EEPROM - EPROM elettricamente**
 - come una PROM ma, per essere riprogrammata, è sufficiente applicare un impulso elettrico.
- **Memoria flash**
 - tipo speciale di EEPROM che è cancellabile a blocchi e riscrivibile.

Chip della CPU

Tutte le CPU moderne sono contenute in un unico chip che possiede un **insieme di pin e segnali** attraverso i quali la CPU comunica con il mondo esterno.



I pin di una CPU possono essere divisi in tre categorie:

- **Indirizzi**
- **Dati**
- **Controllo**

Questi pin sono connessi agli altri componenti (memoria, I/O) tramite bus.

Quando la CPU esegue un'istruzione:

- pone l'indirizzo di memoria dove si trova l'istruzione sul bus indirizzi;
- informa la memoria che vuole eseguire un'operazione di lettura tramite le linee di controllo;
- la memoria risponde ponendo la parola sul bus dati e asserendo un segnale per indicare che l'operazione è stata svolta;
- la CPU riceve questo segnale di controllo, legge i dati e si predisponde per eseguire l'istruzione.

I principali parametri che determinano le **prestazioni di una CPU** sono il **numero di pin per l'indirizzamento e per i dati**:

Un chip con m pin d'indirizzo può indirizzare fino a 2^m locazioni di memoria;

Un chip con n pin per i dati può leggere o scrivere una parola di n bit in una singola operazione.

Oltre a questi pin, le CPU sono dotate di **pin di controllo**, i quali possono regolare il flusso e la temporizzazione dei dati da e verso la CPU. Generalmente, i pin di controllo possono essere raggruppati nelle seguenti categorie:

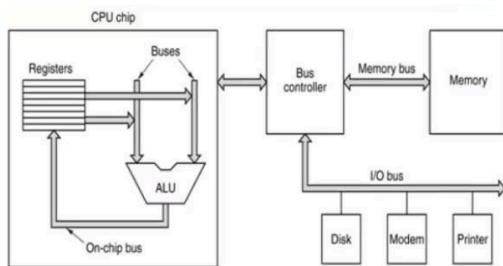
controllo del bus;

interrupt → input che giungono alla CPU dalle periferiche;
 arbitraggio del bus → per regolare il traffico sul bus;
 comunicazione col coprocessore;
 stato;
 altro.

Bus di un calcolatore

Un **bus** è un collegamento elettrico che unisce diversi dispositivi.

I bus possono essere classificati in base alla loro funzione: alcuni sono interni alla CPU (pensiamo al datapath), altri esterni per collegarla alla memoria o ai dispositivi I/O.



I primi PC avevano un unico bus esterno chiamato **bus di sistema**. Era composto da 50-100 fili paralleli di rame che si inserivano nella scheda madre.

Mentre i progettisti di CPU possono utilizzare il tipo di bus che desiderano, per i bus esterni bisogna invece definire delle precise regole di funzionamento. Tale insieme di regole è detto **protocollo del bus**.

Funzionamento di un bus

I dispositivi **master (o attivi)** sono quei dispositivi pronti per il trasferimento dati.

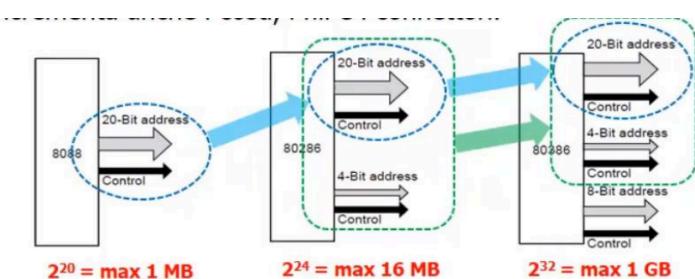
I dispositivi **slave (o passivi)** sono quei dispositivi in attesa di una risposta.

Spesso i segnali digitali generati dalle periferiche sono troppo deboli per alimentare un bus. Per tale motivo, molti master sono connessi al bus tramite un chip chiamato **driver del bus**, che funge da amplificatore digitale; analogamente avviene per i slave, che sono connessi al bus tramite un **ricevitore del bus**.

Aampiezza del bus

Si può intuire che, **maggior è il numero di linee d'indirizzo di un bus, maggiore sarà la quantità di memoria che la CPU potrà indirizzare direttamente**. Se un bus ha n linee d'indirizzi, una CPU può indirizzare fino a 2^n locazioni di memoria.

Nonostante ciò, avere bus più larghi comporta un utilizzo maggiore di fili, di conseguenza una maggiore occupazione di spazio e la necessità di utilizzare connettori più grandi.



Esistono dei modi per aumentare la larghezza di banda dei dati su un bus:

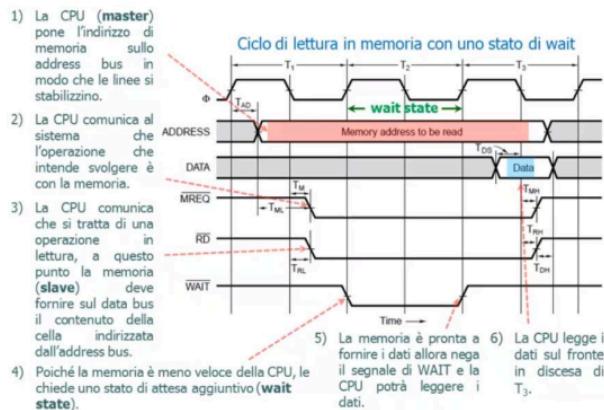
- **diminuire il periodo di clock del bus** (quindi più trasferimenti al secondo);

- ciò comporta difficoltà in termini di **disallineamento del bus** (i segnali su linee diverse viaggiano a velocità differenti) e per la **perdita di retrocompatibilità**
- **aumentare la larghezza dei dati del bus.**
 - per risolvere il problema di bus troppo ampi, i progettisti optano per un bus multiplexato, avendo di conseguenza un rallentamento del sistema.

Temporalizzazione del bus

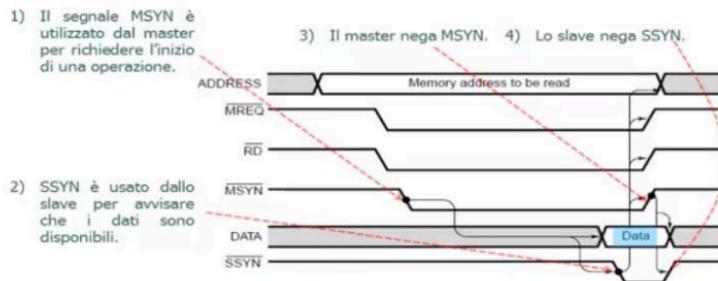
In termini di temporalizzazione, esistono due differenti approcci: **bus sincrono e asincrono**. A differenza dei bus asincroni, i bus sincroni utilizzano un clock che determina la temporizzazione delle attività sul bus (**ciclo di bus**): ogni operazione richiede un numero di periodi di clock per essere eseguita.

Bus Aincrono



- A partire dal fronte di salita di T_1 , la CPU fornisce l'indirizzo della parola sulle linee d'indirizzo;
- Dopo che le linee d'indirizzo si sono stabilizzate sui nuovi valori, vengono assegnati MREQ e RD → Si sta accedendo in memoria in lettura;
- Poiché la memoria è meno veloce della CPU, la memoria assegna WAIT per segnalare alla CPU di non aspettarla;
- Dopo un lasso di tempo, la memoria è pronta per fornire i dati → nega WAIT e inserisce sulle linee appropriate i dati;
- La CPU legge le linee dei dati, memorizzando il valore in un suo registro. Dovendo leggere i dati, la CPU nega MREQ e RD

Bus Asincrono



- Il master del bus, dopo aver assegnato MREQ e RD, assegna uno speciale segnale chiamato MSYN (Master SYNchronization);
- Quando lo slave vede questo segnale, esegue il lavoro richiesto alla massima velocità possibile;
- Una volta terminato il proprio compito, assegna SSYN (Slave SYNchronization);
- Quando il master vede che SSYN è stato assegnato, sa che i dati sono disponibili e può quindi memorizzarli;

- Successivamente, il master nega le linee d'indirizzo, MREQ, RD e MSYN;
- Lo slave, vedendo la negazione di MSYN, nega SSYN.

Un insieme di segnali che coordina in questo modo due dispositivi con lo scopo di non farli interferire è detto **full handshake**.

Arbitraggio del bus

L'arbitraggio del bus è utilizzato per prevenire situazioni di conflitto in cui due o più dispositivi tentano di diventare, nello stesso momento, master del bus.

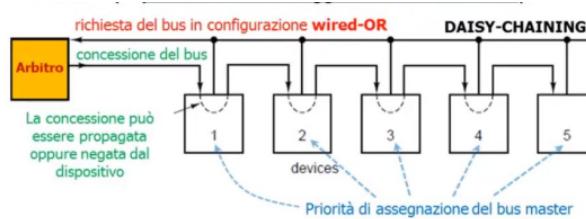
L'arbitraggio può essere **centralizzato** o **decentralizzato**

Arbitraggio centralizzato

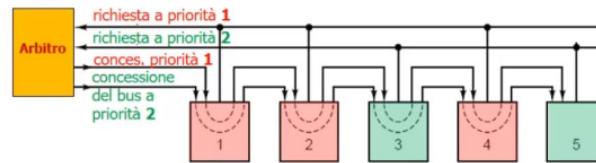
In questo schema un singolo arbitro del bus determina chi sarà il prossimo master. Inoltre CPU l'arbitro è integrato nel chip stesso della CPU.

Il bus contiene un'unica linea di richiesta OR-cablaglio che in qualsiasi momento può essere asserita da uno o più dispositivi.

Quando l'arbitro vede una richiesta di utilizzo del bus, lo concede assegnando la linea per la concessione del bus. Quando il dispositivo fisicamente più vicino all'arbitro vede la concessione, effettua un controllo per verificare se ne ha fatto richiesta. In caso affermativo, si impossessa del bus, negandola a tutti i suoi successori; in caso contrario, propaga la concessione nei dispositivi successivi, finché un dispositivo accetterà la concessione e si impossesserà del bus. Questo schema è detto **collegamento a festone**: assegna ai dispositivi una proprietà in base alla vicinanza all'arbitro.



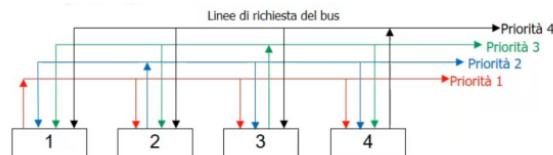
Un'altra tipologia consiste nel definire dei livelli di priorità: per ciascun livello, esiste una linea per effettuare la richiesta e una per segnalare la concessione. Se due dispositivi di priorità diversa richiedono il bus, l'arbitro assegnerà al dispositivo con priorità elevata.



Arbitro Decentralizzato

In questo scenario, ogni dispositivo possiede una propria linea di richiesta ed una di priorità. Quando un dispositivo vuole utilizzare il bus, verifica che non ci sia una richiesta con priorità più alta. Al termine di utilizzo del bus, la linea di richiesta deve essere negata.

Lo svantaggio rilevante consiste in troppi collegamenti.



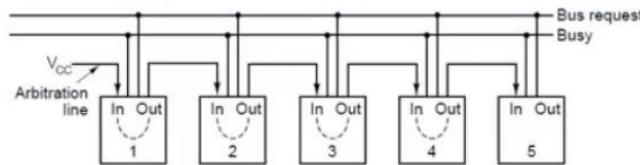
Un'altra tipologia di schema utilizza tre linee: una per la richiesta del bus, una linea per indicare che il bus è occupato, una linea di arbitraggio propagata tra i dispositivi in cascata (è un collegamento a festone).

Quando nessun dispositivo richiede il bus la linea di arbitraggio, che è asserita, viene propagata attraverso tutti i dispositivi.

Per acquisire il bus, un dispositivo deve prima controllare se il bus è inattivo e se il segnale dell'arbitraggio che sta ricevendo (IN) è asserito.

Se IN è negato, non può diventare master e nega OUT;

Se IN è asserito e il dispositivo vuole il bus, allora nega OUT in modo che il dispositivo successivo veda IN negato e neghi anch'esso il proprio OUT. In questo modo, un solo dispositivo rimarrà con IN asserito e OUT negato; a questo punto diventerà master del bus, asserirà BUSY e PUT e inizierà il proprio trasferimento.

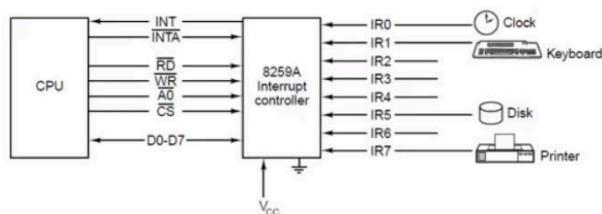


Interrupt handling

Un importante tipo di ciclo di bus serve a **gestire gli interrupt**. Quando la CPU ordina a un dispositivo di I/O di effettuare qualche operazione si aspetta di solito un interrupt alla fine del lavoro; la segnalazione di questi interrupt richiede l'utilizzo del bus

Poiché più dispositivi potrebbero voler generare un interrupt simultaneamente, sorge anche in questo caso la problematica di arbitraggio. Anche in questo caso, la soluzione consiste nell'assegnare priorità ai dispositivi e di utilizzare un arbitro centralizzato per assegnare priorità al dispositivo.

Nell'immagine è presente un controllore di interrupt 8259A. Tale chip ha 8 input IRx (richiesta di interrupt), con i quali si possono collegare direttamente fino a 8 controllori di I/O.



Quando uno di questi dispositivi vuole causare un interrupt, asserisce la propria linea di input.

A questo punto, il controllore asserisce INT, che pilota direttamente il pin di interrupt sulla CPU. Quando la CPU è in grado di gestire l'interrupt, risponde asserendo INTA (conferma dell'interrupt); Ricevuto, il controllore comunica alla CPU quale input ha causato l'interrupt, generando in output il suo numero sul bus dei dati. Quel numero verrà utilizzato come indice per accedere al **vettore di interruzione** e trovare l'indirizzo della procedura da eseguire per poter gestire l'interrupt.

Esempi di CPU

Pentium 4

Il **Pentium 4** è una linea di microprocessori sviluppata da **Intel**, lanciata sul mercato per la prima volta nel **novembre del 2000**. È stato il successore del Pentium III ed è stato basato inizialmente sull'architettura **NetBurst**, progettata per raggiungere alte frequenze di clock, anche a scapito dell'efficienza energetica.

Una delle caratteristiche principali del Pentium 4 era la **pipeline estremamente lunga**, che nelle versioni successive arrivava fino a **31 stadi**. Questo design permetteva teoricamente di raggiungere frequenze molto elevate, ma nella pratica si rivelò poco efficiente: la pipeline lunga aumentava la latenza e rendeva il processore più sensibile agli errori di previsione nei salti condizionali (branch misprediction), rallentando il flusso di istruzioni.

A seconda del modello, il Pentium 4 poteva avere **2 o 3 livelli di cache**:

- Tutti i modelli hanno 8 KB di SRAM (RAM Statica) sul chip per la cache di primo livello L1;
- Il secondo livello di cache L2, condivisa tra dati ed istruzioni, era in grado di memorizzare fino a 1 MB;
- Un'ulteriore versione del Pentium 4 aveva un'ulteriore livello di cache (cache di terzo livello L3), con una dimensione di 2 MB.

Nei sistemi multiprocessore, dove più CPU condividono la stessa memoria principale, è fondamentale garantire che tutte le cache mantengano una visione coerente dei dati. Questo obiettivo viene raggiunto attraverso protocolli di coerenza della cache, tra cui il **snooping** è uno dei più utilizzati.

Lo **snooping** è un meccanismo in cui ogni cache monitora continuamente (o "spia") le comunicazioni sul bus di sistema per intercettare operazioni di lettura o scrittura relative a indirizzi di memoria che potrebbe avere

memorizzato. In pratica, ogni cache osserva il traffico del bus per rilevare se altri processori stanno accedendo a dati che essa stessa possiede.

Il compito principale dello snooping consiste nel mantenere la **coerenza dei dati**.

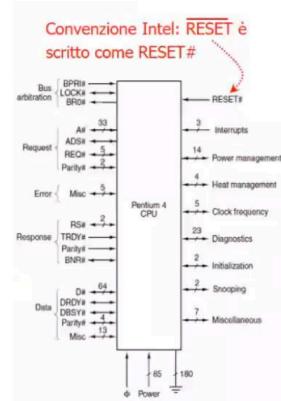
Il Pentium 4 era noto principalmente per raggiungere temperature elevate. Come sappiamo, qualsiasi dispositivo elettronico che produce molto calore deve assorbire molta energia, e in un computer portatile utilizzare troppa energia non è desiderabile perché si consuma presto la carica della batteria.

Intel ha quindi creato un sistema per **"addormentare"** la **CPU** quando non serve: quando la CPU è inattiva (cioè non sta eseguendo istruzioni), entra in uno stato detto **idle**. Se l'inattività si prolunga, la CPU entra in uno stato ancora più profondo di risparmio energetico, chiamato **sleep o deep sleep**.

Il ciclo idle rappresenta lo "stato di quiete" del computer, quindi se non stai facendo nulla si attesta su valori del 98 o 99% (cioè la CPU è impegnata all' 1 o 2%), e non occupa il processore.

Il Pentium 4 ha 478 pin:

- 198 per i segnali;
- 85 per l'alimentazione;
- 180 per la massa;
- 15 liberi per usi futuri



Intel Core i7

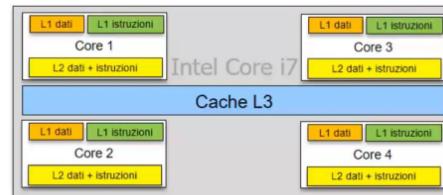
Intel Core i7 è una famiglia di processori di fascia alta sviluppata da **Intel**, introdotta nel **2008** con l'architettura **Nehalem**. È pensata per offrire elevate prestazioni in applicazioni come gaming, produttività, editing video e multitasking pesante.

Caratteristiche più rilevanti:

- Supporta l'**hyperthreaded** → ogni core fisico può gestire **più thread**, raddoppiando la capacità di calcolo in alcune operazioni.
- Dispone di 3 livelli di cache e ciascun core effettua uno snooping sul bus di collegamento con la memoria per garantire la consistenza delle informazioni.

L'immagine descrive i livelli di cache dell'Intel Core i7:

- Due cache di livello L1 distinte per dati e istruzioni, per ogni core;
- Una cache L2 per dati e istruzioni, per ogni core;
- Una cache L3 condivisa tra i core.



UltraSPARC III

L'**UltraSPARC III** è un microprocessore sviluppato da **Sun Microsystems** (poi acquisita da Oracle) e basato sull'architettura **SPARC V9 a 64 bit**, progettata per server e workstation ad alte prestazioni.

È stato introdotto nel **2000** ed è stato una pietra miliare nell'evoluzione delle CPU RISC (Reduced Instruction Set Computer).

Era dotata di:

- 2 cache L1, distinte per dati e istruzioni;

- 2 cache L2, una per il prefetche e una per collezionare le scritture, migliorando così l'utilizzo della banda.

La gestione della cache di secondo livello (L2) presenta una distinzione chiave tra i componenti interni ed esterni al chip

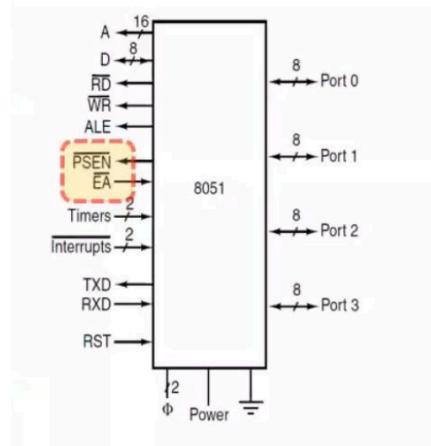
- Il controller della cache e la logica di ricerca dei blocchi di cache era all'**interno** del chip. Questa integrazione consente al controller di operare alla stessa velocità del processore, migliorando l'efficienza nella gestione dei dati e riducendo la latenza nell'accesso alle informazioni memorizzate nella cache.
- La memoria della cache era **esterna**. Questo permise ai progettisti di scegliere la memoria con il miglior rapporto qualità-prezzo senza essere vincolati dalla CPU.

Per connettere la CPU a più memorie, la Sun Microsystems sviluppò l'architettura **UPA**: è un'architettura di interconnessione ad alta velocità che permette una **comunicazione diretta** e veloce tra la CPU e altri componenti, minimizzando il collo di bottiglia che c'era nei sistemi precedenti con bus condivisi.

Il microcontrollore 8051

L'**8051** è uno dei microcontrollori più conosciuti e utilizzati nella storia dell'elettronica. Fu sviluppato da **Intel** nel 1980 ed è diventato uno **standard industriale**.

È un circuito integrato da 40 pin, con 16 bit di address e 8 bit per il bus dati.



Esempi di bus

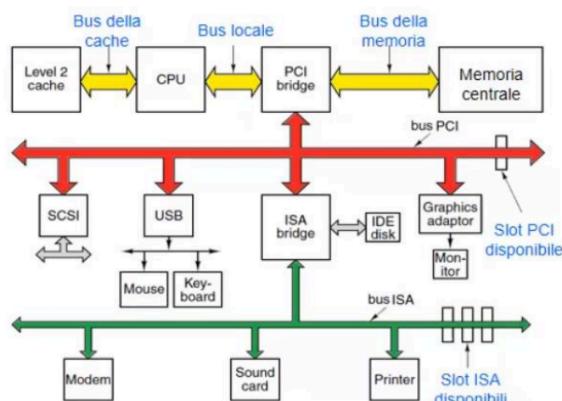
Bus PCI

Il **bus PCI** nasce in sostituzione con i vecchi bus **EISA**: con l'introduzione dei giochi multimediali, la velocità dei bus EISA risulta insufficiente. Pertanto, si progetta un nuovo bus con una larghezza di banda molto più ampia rispetto a EISA.

Nonostante abbia una velocità apprezzabile, rimasero due problematiche:

- La larghezza di banda non era sufficiente per essere usata come bus di memoria;
- Non era compatibile con tutte le vecchie schede ISA ancora in circolazione.

La soluzione intrapresa da Intel consiste nel progettare calcolatori con **tre o più bus**. Come mostrato in figura, la CPU può comunicare con la memoria tramite uno speciale bus di memoria; inoltre, è presente un bus ISA **collegato al bus PCI tramite il bridge ISA**.



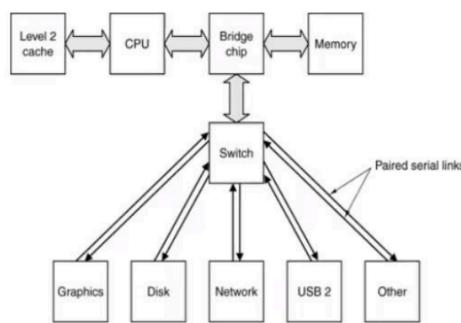
Il **bridge ISA** e il **bridge PCI** sono componenti utilizzati nei computer per consentire la comunicazione tra diverse tipologie di bus. Il bridge PCI connette il bus PCI con la memoria e la CPU, mentre il bridge ISA rende possibile la comunicazione tra il bus ISA e il bus PCI.

Il vantaggio principale di quest'architettura è che la CPU, usando un bus di memoria proprietario, ha una larghezza di banda estremamente alta verso la memoria. Il bus PCI offre un'ampia larghezza di banda per periferiche; con il bus ISA, invece, è ancora possibile utilizzare le vecchie schede.

Bus PCI express

Il bus PCI ha un problema: col tempo si sono sviluppati dispositivi di I/O la cui velocità è eccessiva per questa tipologia di bus. La soluzione proposta da Intel è proprio il **bus PCI express**.

La caratteristica fondamentale di quest'architettura è la **rinuncia al bus parallelo e un utilizzo di un'architettura basata su connessioni punto-a-punto ad alta velocità**.



Ciò che fa PCI express è fornire un **commutatore di uso generale** per connettere i chip mediante collegamenti seriali. Come mostrato in figura, ogni chip di I/O ha una connessione dedicata punto-a-punto con il commutatore, che consiste in una coppia di canali unidirezionali, uno che giunge al commutatore e uno che parte da esso.

Si possono analizzare tre aspetti fondamentali che differisce il bus PCI express dal PCI:

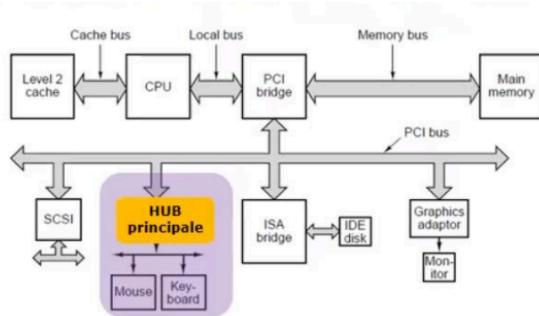
- un commutatore centralizzato al posto di un bus;
- uso di strette connessioni seriali punto-a-punto al posto di un bus parallelo;
- il modello concettuale del bus PCI è quello di un master che lancia allo slave un comando per leggere una parola oppure un blocco di parole. Il modello del bus PCI express è quello di un dispositivo che spedisce **un pacchetto di dati** a un altro dispositivo.

Il concetto di **pacchetto** consiste in un'intestazione e in un campo dati; l'intestazione contiene le informazioni di controllo, il campo dati contiene i dati che devono essere trasferiti.

USB

L'**USB** (Universal Serial Bus) è uno standard di connessione utilizzato per collegare dispositivi lenti come tastiere, mouse, stampanti, dischi rigidi esterni, smartphone e molti altri dispositivi a un computer o a un altro dispositivo elettronico. È stato introdotto nel 1996 e ha evoluto nel tempo, diventando uno dei metodi di connessione più comuni e versatili.

Un sistema USB si compone di un **hub principale** che si collega al bus del sistema e che possiede delle prese per i cavi che connettono i dispositivi di I/O o per l'hub di espansione. Quindi, la topologia di un sistema USB è un albero cui radice è l'hub principale.



Il cavo consiste in 4 collegamenti:

- due per i dati;
- uno per l'alimentazione;
- uno per la terra.

Quando viene collegato un nuovo dispositivo, l'hub principale rileva l'evento e interrompe il sistema operativo; quest'ultimo interroga il dispositivo per sapere di che periferica si tratta e di quanta banda ha bisogno:

Se SO decide che c'è una sufficiente larghezza di banda, gli assegna un indirizzo univoco e scarica nel dispositivo altre informazioni necessarie a configurare i registri.

L'hub effettua un collegamento punto-punto con i dispositivi di I/O come se ci fossero dei tubi. Per mantenere il sincronismo, l'hub ogni ms spedisce in broadcast un nuovo **frame**.

Un frame è associato a un condotto di bit e consiste in pacchetti, il primo dei quali viaggia dall'hub principale verso il dispositivo.

USB supporta 4 tipologie di frame:

- **controllo** → usati per configurare il dispositivo, inviargli i comandi, interrogarlo sullo stato;
- **isocroni** → usati per dispositivi real-time che necessitano di spedire o accettare dati ad intervalli di tempo precisi;
- **bulk** → usati per grandi trasferimenti di dati come nel caso delle stampanti;
- **interrupt** → fondamentali in quanto USB non supporta il concetto di interruzione; senza di essi, SO sarebbe costretto ad interrogare in polling il dispositivo.

Un frame contiene uno o più **pacchetti**:

- **token** → usati per il controllo del sistema dall'hub al device;
- **dati** → per la sincronizzazione;
- **handshake**;
- **speciali**.

