FINAL

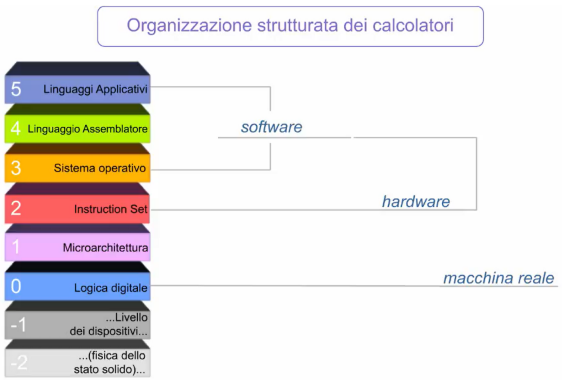
ORGANIZZAZIONE STRUTTURATA DEI CALCOLATORI

I calcolatori sono progettati come un insieme di livelli, ognuno dei quali si basa sui livelli precedenti. Esiste quindi una gerarchia partendo dal livello hardware di base e costruendoci sopra una serie di astrazioni, strutture dati e funzionalità diverse per fornire a noi utenti una visione più elevata della macchina.

L'insieme di tipi di dati, operazioni e caratteristiche di ogni livello prende il nome di **architettura**. Tutto ciò che sta dentro il sistema operativo sarà eseguibile solo su calcolatori con la stessa architettura di sistema.

- **Microarchitettura (1):** Composto da circuiti sequenziali o circuiti combinatori (cioè solo con porte logiche) che rappresentano funzionalità più complesse (somma, sottrazione, etc.). Può essere diviso in due parti principali:
  - Elaborazione (datapath): Legge dati dai registri, esegue operazioni aritmetico-logiche e scrive risultati in altri registri.
  - Controllo: Legge l'istruzione, definisce le operazioni e informa l'ALU. Per leggere l'istruzione che dovrà essere eseguita ha bisogno di un registro dove memorizzarla temporaneamente (Program Counter) e ha bisogno di mantenere l'informazione relativa a quale istruzione all'interno del programma da eseguire è attualmente in esecuzione (Instruction Register). L'elemento che compone ciò è la CU.

- **Instruction Set (2):** Livello cui si fa riferimento quando si parla di linguaggio macchina. È composto da un insieme di istruzioni che possono essere comprese dalla microarchitettura, e la microarchitettura funziona da interprete per l'ISA.



- **Sistema Operativo (3):** Livello ibrido che può: essere tradotto in L2, essere interpretato parzialmente in L1 o eseguito direttamente in L0. È un livello che

comprende tutte le istruzioni del livello 2 insieme ad altre istruzioni. Permette di vedere la memoria in modo semplificato e di eseguire più programmi in contemporanea.

I livelli bassi (1,2,3) supportano il funzionamento dei compilatori e degli interpreti utilizzati ai livelli alti (4,5). I livelli bassi sono linguaggi numerici (binari o esadecimale) mentre i livelli alti sono linguaggi composti da parole (più vicini a noi umani).

- **Linguaggio assemblatore (4):** fornisce una rappresentazione simbolica del linguaggio del sistema operativo. La traduzione avviene in forma compilata attraverso un programma che viene chiamato assemblatore, che traduce il linguaggio assemblatore in linguaggio binario.
- **Linguaggi applicativi (5):** Linguaggi di alto livello che vengono tradotti nel linguaggio assemblatore, poi parzialmente tradotti nel sistema operativo fino ad arrivare alla logica digitale. Il compilatore attraversa tutti i livelli che abbiamo visto in precedenza.

DATAPATH

Microarchitettura: strato del calcolatore che si appoggia sul livello logico e interpreta le istruzioni definite dall'insieme delle istruzioni eseguibili da quel processore. Tale insieme di istruzioni che il processore è in grado di eseguire è detto ISA (Instruction Set Architecture). Non esistono architetture standard: esistono famiglie di architetture.

Il livello di microarchitettura prevede le seguenti quattro fasi:

- Un microprogramma che è presente nella memoria ROM: un insieme di microistruzioni (che controllano il datapath) che il processore è in grado di eseguire.
- **FETCH:** lettura delle singole istruzioni del programma che deve essere eseguito (trasferisce le istruzioni del livello assembler (ISA)). Il processore trasferisce dalla memoria centrale all'interno del processore stesso l'istruzione che deve essere eseguita. Per disporre dell'indirizzo di memoria dove è memorizzata la prossima istruzione da eseguire viene utilizzato il Program Counter.
- **DECODE:** il processore guarda l'istruzione che è stata trasferita tramite la memoria RAM e cerca di riconoscerla, tramite il suo codice operativo (op-code).

- **EXECUTE:** il processore esegue ogni singola istruzione. Per eseguirle, il processore gestisce un insieme di variabili. L'insieme delle variabili definisce lo stato del processore. Ogni variabile è memorizzata in un registro il quale prevede l'utilizzo di un insieme di bistabili che sono montati sullo stesso chip del processore. Sullo stesso processore vi è poi un opportuno registro che tiene traccia dell'istruzione eseguita (PC). Il Program Counter definisce la cella di memoria dove è memorizzata la prossima istruzione che deve essere eseguita.

Datapath: è quella parte di CPU che contiene: l'ALU, i suoi ingressi e le sue uscite e i vari registri. I primi componenti che incontriamo all'interno del datapath sono i registri di controllo della memoria:

- **MAR (Memory Address Register):** viene utilizzato per specificare qual è l'indirizzo della cella di memoria a cui si vuole fare riferimento.
- **MDR (Memory Data Register):** è il registro che contiene il dato, ovvero la parola di memoria. Questo registro conterrà la parola di memoria specificata dal registro MAR in caso di operazione di lettura. In caso di scrittura, il registro MDR conterrà la parola che deve essere memorizzata nell'indirizzo contenuto nel MAR.
- **PC (Program Counter):** Indica la parola di memoria dove è memorizzata la prossima istruzione.
- **MBR:** Viene utilizzato per trasferire dalla memoria gli 8 bit relativi alla parola specificata nel registro PC.

Sincronizzazione nel datapath: La sincronizzazione è gestita dal segnale di clock e quindi dall'onda quadra generata. Tutti i cicli di clock hanno la stessa durata. I valori sono letti sul fronte di discesa del ciclo di clock. I valori sono scritti sul fronte di salita del ciclo di clock.

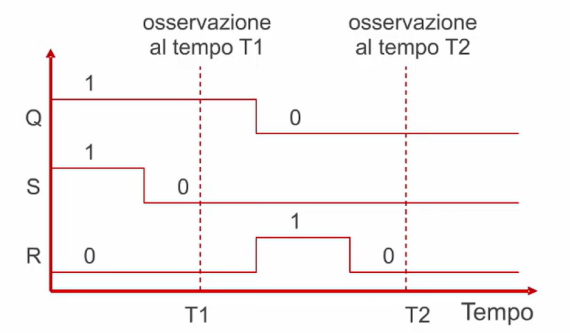
Control Store (CS): Memoria a sola lettura che include tutte le microistruzioni. Per ogni istruzione a livello ISA, il Control Store definisce quali sono le relative microistruzioni che devono essere eseguite per effettuare la fase di fetch code execute di ognuna delle istruzioni definite.

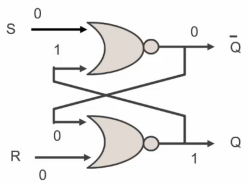
Se voglio fare più operazioni in maniera parallela uso la PIPELINE: permette la parallelizzazione dell'istruzione e di aumentare il throughput.

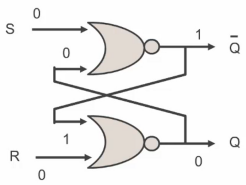
BISTABILI

Bistabile SR: (il nome SR deriva dal fatto che sono presenti due ingressi S (set) e R (reset)). Circuito digitale NON COMBINATORIO dotato di due ingressi S e R e di

un’uscita Q.
Q vale 1 quando S = 1 (S è attivo).
Q vale 0 quando R = 1 (R è attivo).
Nel mentre, Q non subisce variazioni e in più non accade mai che S = R = 1 (sia S che R sono attivi).
Quando S = R = 0 (sia S che R sono inattivi), Q può valere sia 0 che 1, dipende dalla successione di eventi che si sono verificati in precedenza.







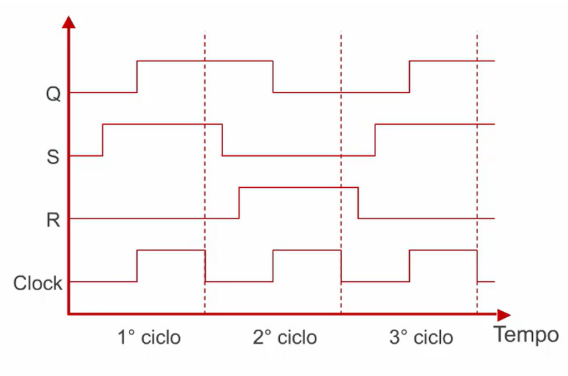
Collegando due porte logiche di tipo NOR in modo retroazionato, si può simulare il funzionamento di un bistabile SR.
Non si tratta di una rete combinatoria, perché quando S = R = 0, Q può valere 1 o 0 e questo non è un comportamento da rete combinatoria.

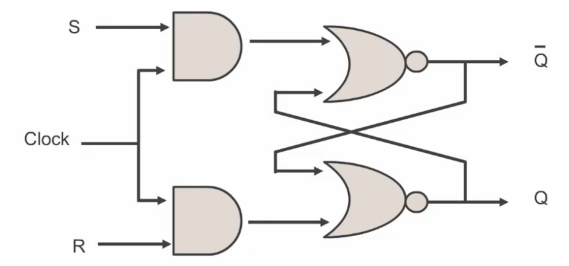
Il bistabile SR è in grado di memorizzare due valori distinti: se Q=1 il bistabile memorizza il bit 1. se Q=0 il bistabile memorizza il bit 0.

Bistabile sincronizzato (con clock)

Per eseguire operazioni (talvolta anche semplici) vi è bisogno di memorizzare i valori assunti negli attimi precedenti. Viene introdotto quindi il **segnale di clock:** un segnale binario che assume 0 e 1 con un andamento periodico nel tempo. Non è altro che una successione di impulsi dove ogni impulso ha larghezza costante e la distanza tra gli impulsi è anch’essa costante.

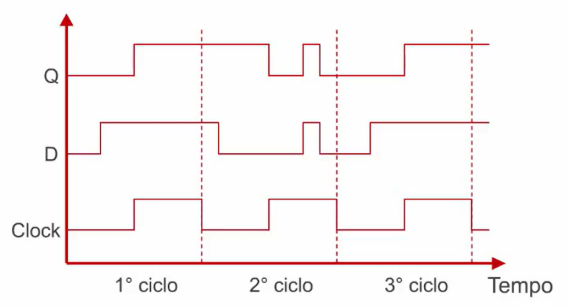
Se il segnale di clock vale 0, il bistabile si limita a mantenere il suo stato corrente.
Se il segnale di clock vale 1, siamo letteralmente nel caso di prima del bistabile SR

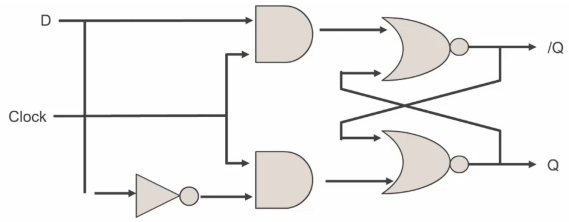




Le due porte AND si comportano come dei semafori lasciando passare il segnale o bloccandolo.

Molto spesso occorre però un bistabile dotato di un ingresso, che si limiti a memorizzare il bit presente sul suo unico ingresso: il **bistabile D sincronizzato (D latch).**





Se il clock vale 0, l'ingresso D non ha alcun effetto e il bistabile mantiene il suo stato corrente.

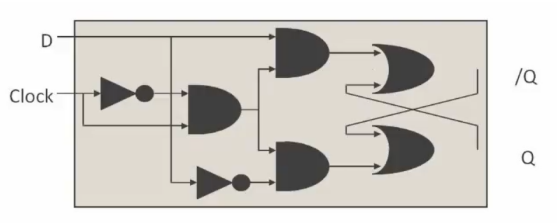
Se il clock vale 1, l'ingresso D è efficace e il bistabile memorizza il valore presente sull'ingresso D.

Lo stato del bistabile può cambiare solo nell'intervallo alto del clock.

Il D latch ha un problema. Quando il clock è attivo, l'ingresso D è efficace: le variazioni di D si propagano immediatamente in uscita Q, la quale può dunque variare più volte. Solo quando il clock torna a 0, Q si stabilizza. È come se, nell'intervallo alto del clock, il bistabile non esercitasse alcuna funzione effettiva di memorizzazione. Questo fenomeno si chiama **trasparenza**.

Esistono due modi per sincronizzare un bistabile con un segnale di clock: la sincronizzazione sul livello (come sopra) e la sincronizzazione sul fronte (come sotto).

Come creare un **flip-flop di tipo D:** Si prende un latch di tipo D e si applica un segnale di clock fortemente asimmetrico. Il duty cycle ha valore infinitesimo causato dal ritardo (anch'esso infinitesimo) della porta not del clock. Quindi, l'ingresso D del latch è efficace solo per un brevissimo intervallo di tempo, che idealmente coincide con l'istante del fronte di salita del clock.



I circuiti sequenziali sono formati da:
- **Bistabili**, di vario tipo, che hanno funzioni di memorizzazione di informazioni, ovvero hanno la funzione di memorizzare dei bit;
- **Porte logiche**, organizzate in reti combinatorie, che hanno funzioni di elaborazione di informazioni.

Il circuito sequenziale ha, in ogni istante, uno stato: i bit memorizzati nei bistabili facenti parte del circuito.

Esistono due famiglie di circuiti sequenziali:

- **Asincroni**, che non fanno uso di clock (es. Bistabile SR): sono poco usati perché difficili da controllare;
- **Sincroni**, cioè che necessitano di clock (es. flip-flop D).

MEMORIA

anco dei registri: sono dei registri strutturati a matrice o vettore, tutti con le stesse dimensioni.

L'unità di base della memoria è chiamato **bit**, può contenere 0 o 1.

Le **memorie** sono composte da celle, ognuna delle quali in grado di memorizzare una parte dell'informazione. Ogni cella ha associato un indirizzo, cioè un numero che definisce come accedere alle informazioni. Tutte le celle della memoria contengono lo stesso numero di bit. La cella è l'unità indirizzabile più piccola della memoria.

Distanza di Hamming: indica il numero di bit diversi che ci sono tra due parole binarie di uguale lunghezza. Il calcolo dei bit differenti può essere eseguito facendo l'OR tra le due parole.

L’accesso alla memoria passa attraverso il bus.

Una gerarchia di memoria è formata da una memoria piccola e veloce (che contiene la copia di alcune celle di quella grande e lenta) e da una grande e lenta. Quando la CPU chiede una particolare risorsa, la richiesta va ad entrambe le celle di memoria: la memoria veloce serve più velocemente la CPU. Se il dato non si trova in quella piccola, la memoria grande recupera il dato e lo manda a quella piccola.

Le memorie piccole e veloci sono chiamate **cache.**

Le cache (sia interne che esterne) sono realizzate con la tecnologia SRAM (Static RAM). La memoria centrale viene realizzata con una tecnologia chiamata DRAM (Dynamic RAM).

RAM Statica: memoria ad accesso casuale che utilizza 2 transistor per bit e può mantenere i dati fin tanto che esiste l’alimentazione.

RAM Dinamica: ogni singolo bit è memorizzato all’interno di un singolo transistor che lavora come un condensatore, ossia mantiene la carica. Però non dura all’infinito quindi è necessario rinfrescare i valori contenuti nei singoli bit per non perdere l’informazione.

Memoria: blocco funzionale di tipo sequenziale complesso. Serve per mantenere a tempo indefinito dati e programmi, e per permetterne l’accesso, in lettura o in scrittura. Ha una struttura a vettore, i cui elementi sono le parole di memoria. Ogni parola di memoria è una sequenza di bit.

Per progettare un chip di memoria bisogna specificare:

- **La capacità**: misurata in numero totale di bit memorizzabili;
- **Le funzioni:** Lettura e scrittura, o solo lettura;
- **Il numero di porte di accesso;**
- **Il tempo necessario per l’accesso:** il tempo che occorre per leggere una parola.

Il contenuto della memoria viene letto o scritto una parola per volta, in un ciclo di clock. Si accede a una parola di memoria tramite la porta di accesso alla memoria. La porta di accesso alla memoria può funzionare in lettura e scrittura, solo in lettura o solo in scrittura.

La porta di accesso alla memoria è formata da:

- **Gli ingressi di indirizzo:** che codificano in binario l’indirizzo della parola su cui si deve operare;

- **Le uscite/ingressi di dato:** che servono per leggere/scrivere una parola;
- **Il comando di lettura/scrittura:** RD = 1 lettura; RD = 0 scrittura;
- **Il comando di abilitazione del componente;**
- **Il comando di abilitazione delle uscite dati.**

Ciclo di lettura:

- **Abilitare il componente** (CS = 1);
- **Inviare l'indirizzo** della parola da leggere;
- **Attivare** il comando di lettura (RD = 1);
- **Non isolare le uscite** (OE = 0).
- Dopo un certo intervallo di tempo il contenuto della parola è disponibile sulle uscite.

Ciclo di scrittura:

- **Abilitare il componente** (CS = 1);
- **Inviare l'indirizzo** della parola da scrivere;
- **Disattivare** il comando di lettura (RD = 0);
- **Isolare le uscite** (OE = 1).
- Dopo un certo intervallo di tempo la parola viene scritta in memoria all'indirizzo indicato.

CPU

Il processore, o CPU (Central Processing Unit), è l'unità funzionale più complessa e più raffinata presente nel calcolatore.

Il processore è un sistema complesso realizzato aggregando numerosi blocchi funzionali combinatori e sequenziali.

La CPU è composta da:

- Registri (memoria): memorizzare dei dati su cui deve operare;
- ALU (Arithmetic Logic Unit): È un blocco funzionale combinatorio (composto da porte logiche, quindi con delle entrate e un'uscita) usato per le operazioni aritmetiche e logiche e non ha retroazioni. Alcune delle operazioni sono: add, sub, compare, multiply, divide, etc.

- CU (Control Unit): gestisce il flusso dei dati, manda avanti gli step.

La CPU ha le seguenti funzioni:

- **Preleva** le istruzioni del programma dalla memoria del calcolatore e le esegue;
- **Legge e scrive** dati dalla e nella memoria centrale del calcolatore, e li elabora, secondo le direttive del programma;
- **Riceve o invia** dati dalle o verso le periferiche, e li elabora, secondo le direttive del programma.

La CPU è realizzata su un unico circuito integrato e la sua interfaccia di comunicazione è completamente definita dalla piedinatura del chip ospitante la CPU. I piedini della CPU sono divisi in tre grandi gruppi: piedini di indirizzo, di dato e di controllo.

Ecco come procede la CPU quando **preleva un'istruzione dalla memoria del calcolatore:**

- **La CPU invia l'indirizzo** della parola di memoria contenente l'istruzione ai piedini di indirizzo;
- **Attiva i piedini di controllo** per comandare alla memoria un'operazione di lettura;
- **La memoria invia la parola richiesta** ai piedini dati della CPU e invia ai piedini di controllo la conferma che il comando è stato eseguito;
- Ricevuta la conferma, **la CPU accetta la parola** ed esegue l'istruzione che questa codifica.

La lettura e la scrittura di dati da o in memoria funzionano in modo del tutto simile al prelievo di un istruzione, idem per la ricezione o l'invio dei dati dalle o alle periferiche.

La CPU non ha altro modo di comunicare con il resto del calcolatore oltre quello appena descritto.

BUS

Il calcolatore è un insieme di unità funzionali: CPU, memoria, unità I/O, etc. Tutte queste unità sono collegate con i **bus**.

I bus si possono dividere in due categorie:

- **Bus interni:** confinati all'interno di una singola unità funzionale, e che collegano i blocchi funzionali contenuti nell'unità stessa. Es. quelli che collegano le parti

interne della CPU (ALU - Registri). Non sono standardizzati e nemmeno resi pubblici;

- **Bus esterni:** si estendono all'esterno dell'unità funzionale e la collegano alle altre unità funzionali. Sono quelli che vediamo quando si apre un calcolatore. Di norma sono standardizzati e pubblici. Si dividono a loro volta in diversi tipi:
  - **Bus di memoria:** collega la CPU e unità funzionali di memoria;
  - **Bus di I/O:** collega la CPU e le unità funzionali di I/O.

Tecniche di realizzazione dei BUS:

- **Schede di Bus** con piste di collegamento e connettori montati sulla scheda;
- **Cavi elettrici** flessibili connettorizzati;
- Alcuni in **fibra ottica;**
- Alcuni con **onde radio.**

In ogni istante, una sola unità funzionale possiede il controllo del Bus, cioè decide quali operazioni di comunicazione eseguire.

L'unità funzionale che in un certo istante detiene il controllo del Bus si chiama **master.** Decide quale operazioni eseguire, lettura oppure scrittura, e in quale istante di tempo. Decide anche qual è l'unità funzionale da leggere oppure da scrivere.

Le rimanenti unità funzionali, che non detengono il controllo del Bus, si chiamano **slave.**

Ogni Bus è controllato da un unico master.
Ogni master può controllare n bus.
Ogni bus può collegare più slave.
In un sistema con n bus ci possono essere massimo n master.
Il calcolatore deve possedere almeno un master (di solito la CPU).
Un calcolatore con un solo bus deve contenere esattamente un master.

Master	Slave	Esempio
CPU	Memoria	Prelievo istruzioni e lettura/scrittura dati
CPU	Unità di I/O	Ricezione/invio dati da/a un'unità di I/O
CPU	Coprocessore	La CPU dà istruzioni al coprocessore
I/O	Memoria	Accesso diretto alla memoria (DMA) *
Coprocessore	CPU	Il coprocessore legge operandi dalla CPU

Per potersi collegare al bus, le unità master e slave sono dotate di un componente di interfaccia, chiamato **bus driver.**

Il bus driver è in grado di: collegarsi e scollegarsi elettricamente al o dal bus, amplificare opportunamente i segnali da trasmettere/ricevere sul/dal bus.

Un bus è diviso in tre componenti:

- **Bus degli indirizzi**
- **Bus dei dati**
- **Bus di controllo**

Si possono unire le funzioni del bus di indirizzo e del bus di dato in un unico gruppo di linee: il **bus multiplato.**

Il funzionamento del bus del calcolatore procede per **cicli di bus**. Esistono due metodi fondamentali per realizzare la scansione dei cicli di bus: il **bus sincrono** e il **bus asincrono**. I due metodi si distinguono per la presenza o meno di un segnale di clock.

BUS SINCRONO

Il bus di controllo contiene una linea che trasporta un segnale di clock a frequenza prestabilita. Il clock viene distribuito a tutte le unità funzionali collegate al bus. Il segnale di clock scandisce le varie transizioni di segnale e il passaggio da un ciclo di bus al ciclo successivo. Tutte le unità sanno sempre quando si passa al ciclo successivo.

BUS ASINCRONO

Non esiste alcun segnale di clock comune alle unità funzionali collegate al bus del calcolatore. Le transizioni di segnale e il passaggio da un ciclo di bus al ciclo successivo non sono sincronizzati. Le unità funzionali osservano il bus di controllo: quando avviene una transizione di segnale significa che si verifica un avanzamento dell’operazione.

Confronti

Il bus sincrono ha il vantaggio di essere semplice da progettare e da controllare.
Il bus sincrono ha lo svantaggio di portare a sprechi di tempo, poiché ogni operazione si deve svolgere in un numero intero di cicli di clock:

- quando un’operazione si potrebbe completare in meno di un ciclo di clock
- quando un’operazione richiede un numero frazionario di cicli di clock

Il bus asincrono ha il vantaggio di essere più efficiente nell'uso dei cicli: l'operazione si completa esattamente nel tempo di cui si abbisogna.
Il bus asincrono ha lo svantaggio di essere complesso da progettare e da controllare.

I bus di calcolatore sono in massima parte di tipo SINCRONO.

Il bus del calcolatore può avere, in ogni istante, un unico master. In caso di cessione del ruolo di master da un'unità funzionale a un'altra, occorre un meccanismo di **arbitraggio del bus** che ne regoli l'utilizzo da parte delle unità funzionali.

Esistono due meccanismi principali di arbitraggio: **centralizzato e distribuito**.

ARBITRAGGIO CENTRALIZZATO

Prevede l'esistenza di un'apposita unità funzionale che svolge la funzione di arbitro del bus. L'arbitro realizza il meccanismo di cessione del controllo del bus, vale a dire del ruolo di master.

Quando l'arbitro riceve una **bus request**, attiva la linea di conferma. La conferma viene passata in cascata alle unità potenziali richiedenti: se un'unità non ha una richiesta pendente, passa la conferma all'unità successiva; altrimenti, trattiene per sé la conferma, senza passarla all'unità successiva, e prende il controllo del bus, comportandosi come master. Questo meccanismo viene chiamato **Daisy Chaining.**

Nei meccanismi di arbitraggio centralizzato, le unità non sono in grado di sapere quando un'unità che aveva chiesto e ottenuto il controllo del bus lo rilascia, rendendolo disponibile per altre unità. Si migliora l'efficienza dell'arbitraggio aggiungendo una terza linea di controllo: il **segnale di accettazione:** un'unità richiede e ottiene dall'arbitro il controllo del bus. Non appena l'arbitro le invia la conferma, l'unità che prende il controllo del bus attiva la linea di accettazione e disattiva la linea di richiesta. Non appena l'arbitro vede attivarsi la linea di accettazione, disattiva la linea di conferma.

ARBITRAGGIO DISTRIBUITO

Prevede una linea di richiesta bus per ogni unità. Non esiste alcun arbitro del bus. Le unità hanno priorità diverse e fissate. Le unità osservano tutte le linee di richiesta del bus.

Quando l'unità deve diventare master, si accerta che nessun'altra unità a priorità superiore alla sua abbia attivato la propria linea di richiesta bus. Se è così, l'unità attiva la sua linea di richiesta bus e ottiene il controllo del bus, diventando master.

Confronto: si risparmia sul costo dell’arbitro, ma il numero di linee di controllo cresce con quello delle unità.