

Introduzione all'architettura dei Calcolatori

Introduzione

- Che cos'è un calcolatore?
- Come funziona un calcolatore?
 - è possibile rispondere a queste domande in molti modi, ciascuno relativo a un diverso punto di vista
 - in questo corso, il punto di vista prevalente è quello del calcolatore come macchina programmabile, ovvero in grado di eseguire programmi

Architettura dei calcolatori

- Che cos'è un calcolatore? Come funziona un calcolatore?
 - un calcolatore è un sistema
 - un **sistema** è un oggetto costituito da molte parti (componenti) che interagiscono, cooperando, al fine di ottenere un certo comportamento
- Studiare l'**architettura** di un sistema vuol dire
 - individuare ciascun componente del sistema
 - comprendere i principi generali di funzionamento di ciascun componente
 - comprendere come i vari componenti interagiscono tra di loro

Hardware e software

- La prima decomposizione di un calcolatore è relativa alle seguenti macro-componenti:

hardware

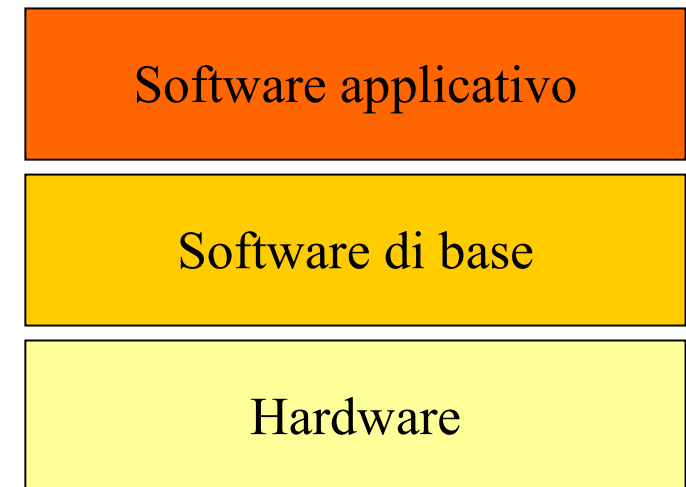
- la struttura fisica del calcolatore, costituita da componenti elettronici ed elettromeccanici

software

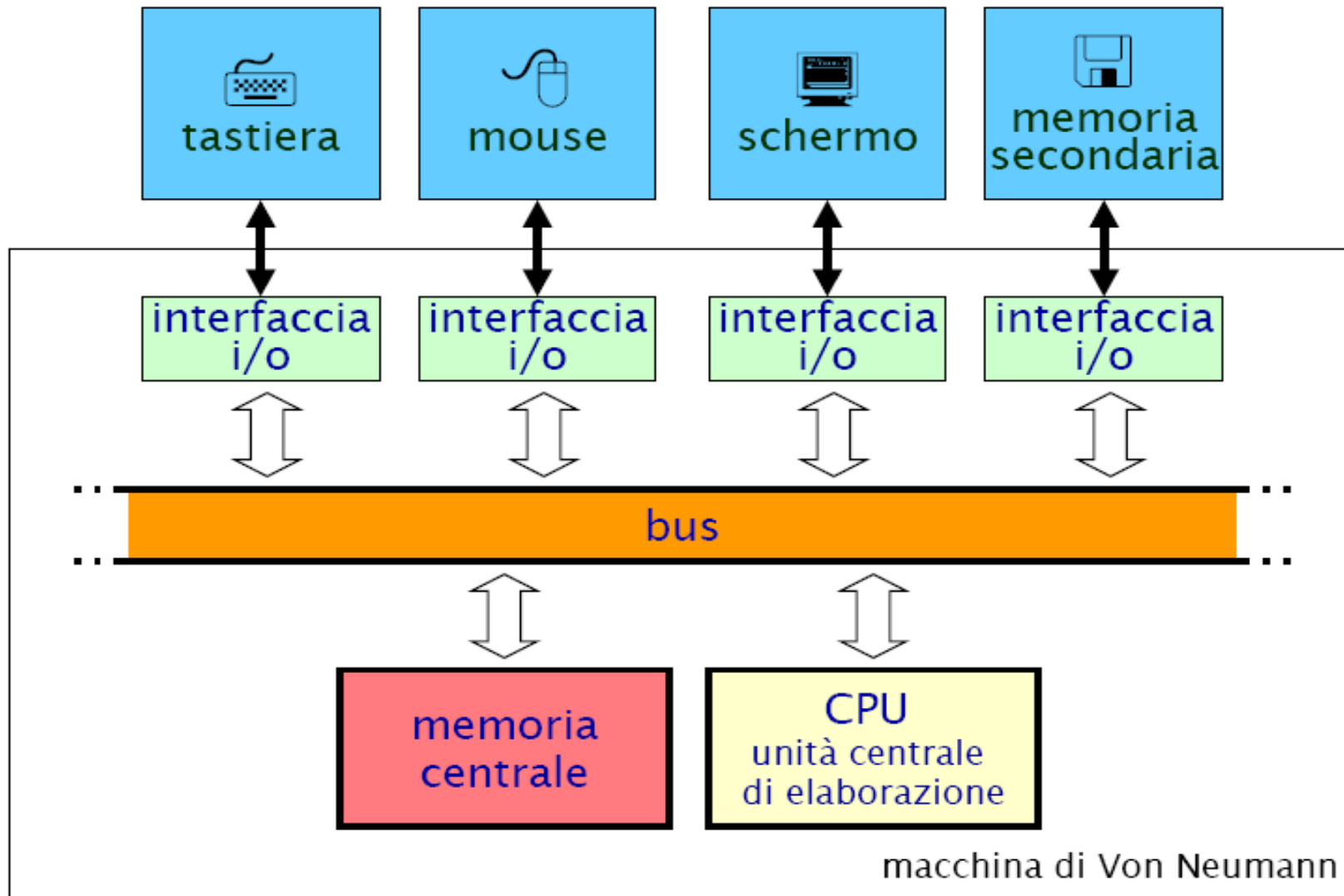
- l'insieme dei programmi che consentono all'hardware di svolgere dei compiti utili
- il software comprende il **software di base** (tra cui il sistema operativo) e il **software applicativo**

Organizzazione a livelli

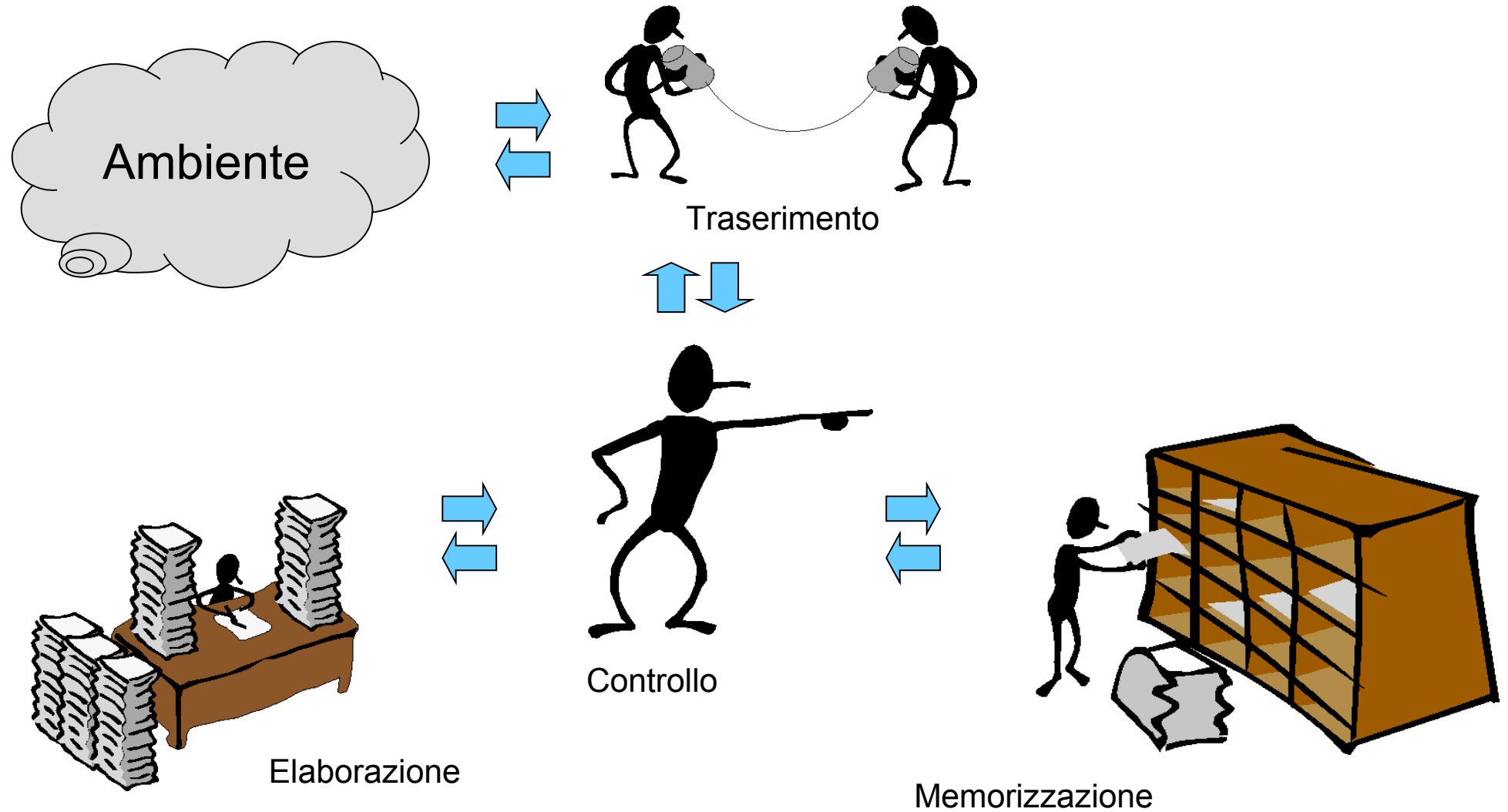
- Hardware e software sono organizzati a **livelli** (o **strati**)
- ciascun livello corrisponde a una macchina (reale o virtuale) in grado di eseguire un proprio insieme di operazioni
- ciascun livello fornisce un insieme di operazioni più semplici da utilizzare rispetto a quelle del livello sottostante
- ciascun livello è realizzato in termini dell'insieme di operazioni fornite dal livello immediatamente sottostante



Componenti di un Calcolatore

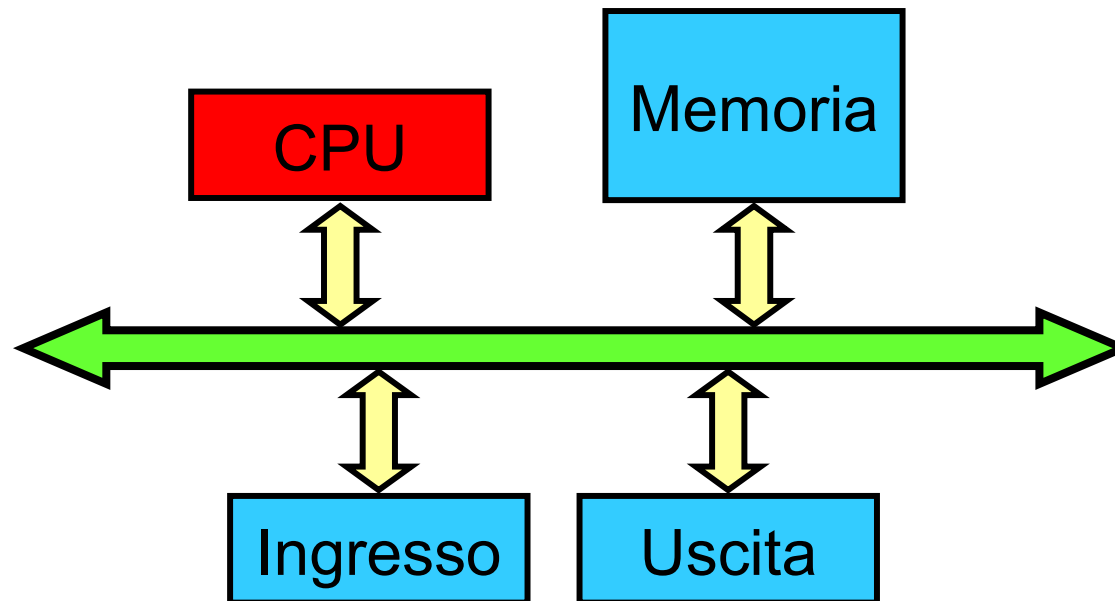


Vista funzionale di un Calcolatore



Bus e Master-Slave

- Il bus è una linea a cui sono contemporaneamente connesse le unità del calcolatore e che consente il trasferimento di dati tra tali unità
 - **Problema:** contesa su un mezzo condiviso!
 - **Soluzione:** CPU = master, periferiche = slave



Bus e Master-Slave - Pregi

- **Semplicità:** 1 sola linea di connessione \forall # di dispositivi
- **Estendibilità:** nuovi dispositivi possono essere aggiunti tramite un'interfaccia al bus senza influenzare l'HW preesistente
- **Standardizzabilità:** definizione di normative che consentono a periferiche di costruttori diversi di interagire correttamente

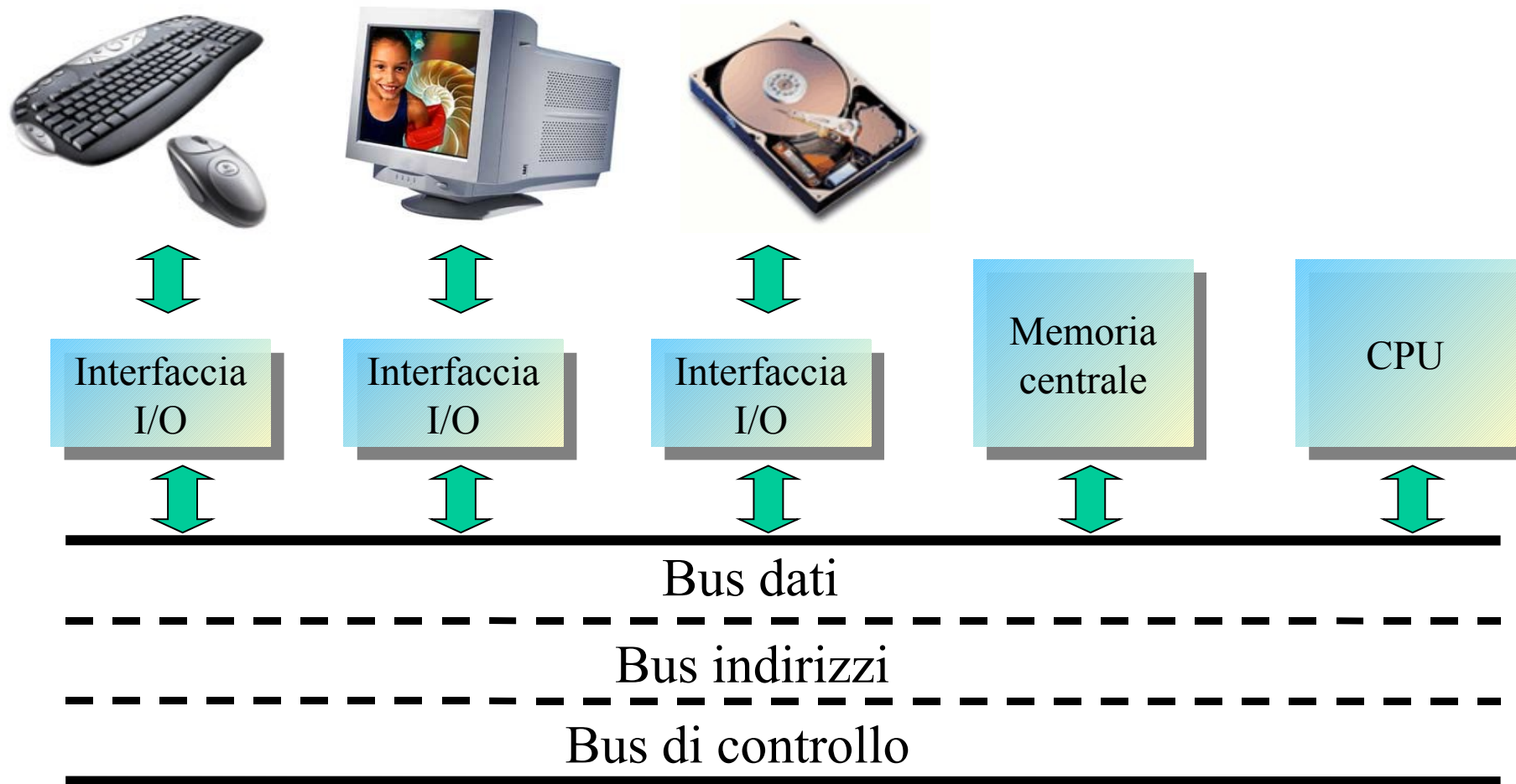
Bus e Master-Slave - Difetti

- **Lentezza:** l'uso in mutua esclusione del bus inibisce almeno parzialmente la parallelizzazione delle operazioni di trasferimento di dati tra dispositivi
- **Limitata capacità:** al crescere del numero di dispositivi la presenza di una sola linea comporta un limite alla capacità di trasferire dati
- **Sovraccarico della CPU:** l'unità centrale viene coinvolta in tutte le operazioni di trasferimento di dati

Tipi di Bus

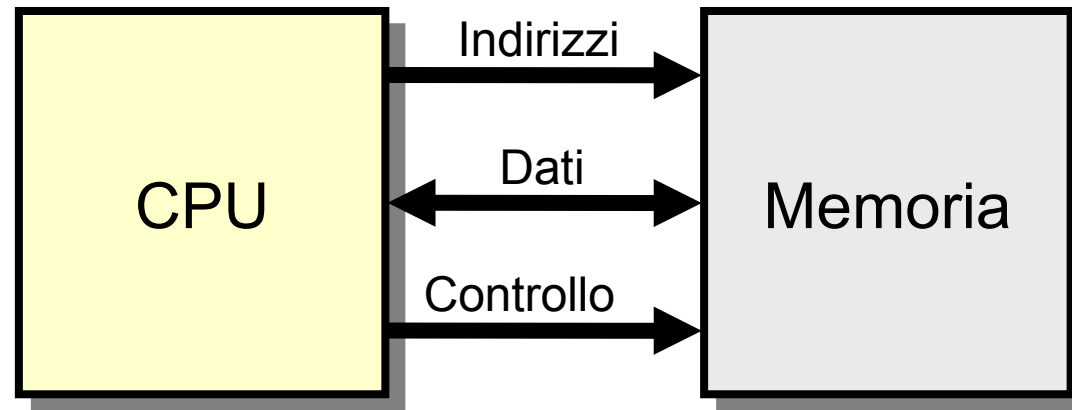
- **Bus dati:** utilizzato per trasferire dati (es. fra memoria e CPU, fra CPU e interfacce di I/O)
- **Bus indirizzi:** che identifica la posizione delle celle di memoria un cui la CPU va a scrivere o leggere
- **Bus di controllo:** in cui transitano i segnali di controllo che consentono di selezionare le unità coinvolte in un trasferimento dati (sorgente e destinazione), di definire la direzione dello scambio (scrittura o lettura)

Lo Schema di Riferimento



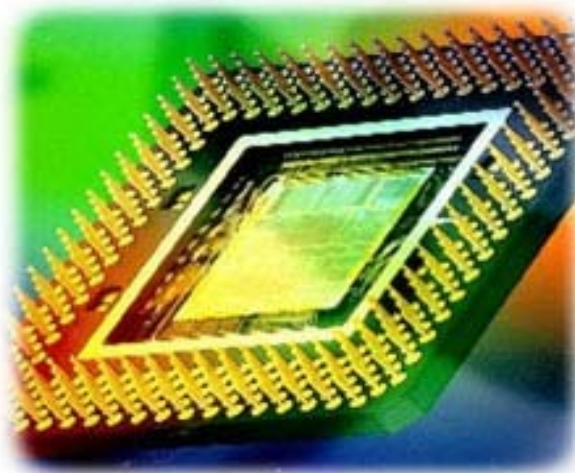
Architettura di Von Neumann

- Burks, Goldstein e Von Neumann sono stati i primi a proporre che il codice del programma potesse essere memorizzato nella stessa memoria dei dati

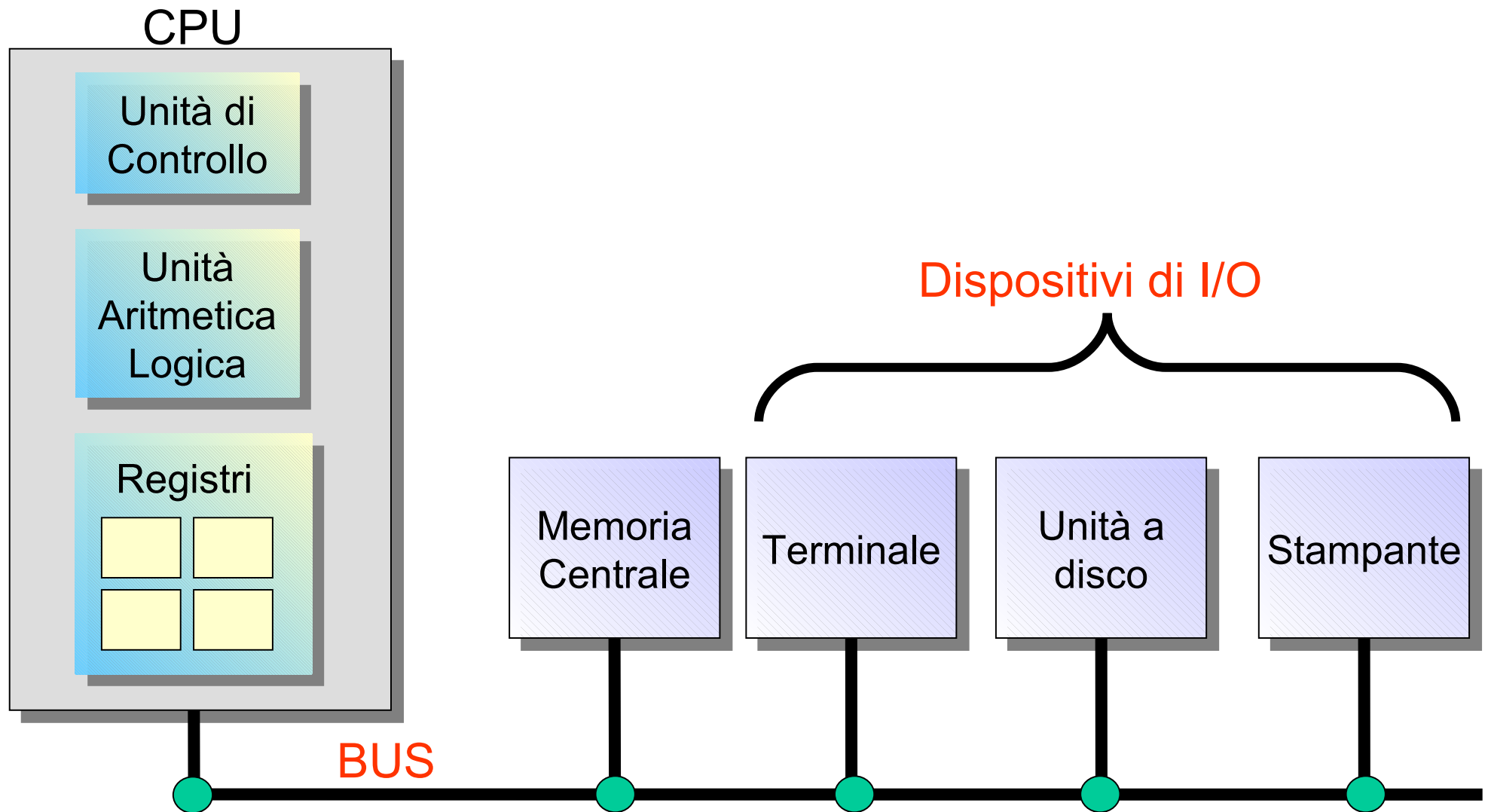


- Memoria indifferenziata per dati o istruzioni
- Solo l'interpretazione da parte di CPU stabilisce se una data configurazione di bit è da riguardarsi come un dato o come un'istruzione

L'Unità Centrale di Elaborazione



Organizzazione Tipica (bus oriented)



Elementi di una CPU

■ Unità di controllo

→ Legge le istruzioni dalla memoria e ne determina il tipo

■ Unità aritmetico-logica

→ Esegue le operazioni necessarie per eseguire le istruzioni

■ Registri

→ Memoria ad alta velocità usata per risultati temporanei

→ Determina il parallelismo della CPU

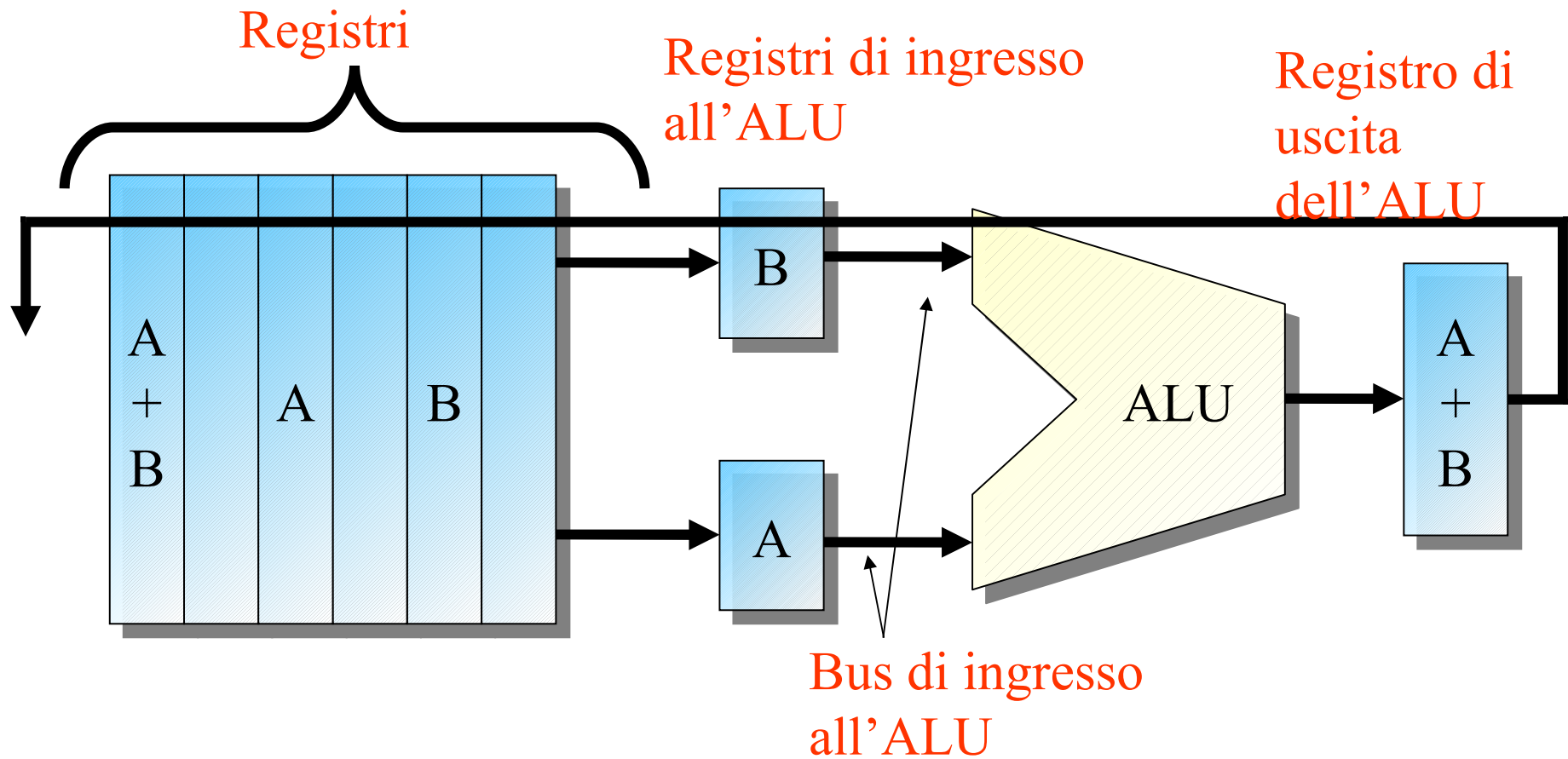
→ Esistono registri generici e registri specifici

- ✓ Program Counter (PC)
- ✓ Instruction Register (IR)
- ✓ ...

Tre Tipologie di Istruzioni

- Istruzioni Aritmetico Logiche (Elaborazione dati)
 - Somma, sottrazione, divisione, ...
 - And, Or, Xor, ...
 - Maggiore, minore, uguale, maggiore uguale, ...
- Controllo del flusso delle istruzioni
 - Sequenza
 - Selezione
 - Ciclo a condizione iniziale, a condizione finale, ...
- Trasferimento di informazione
 - Trasferimento dati e istruzioni tra CPU e memoria
 - Trasferimento dati e istruzioni tra CPU e dispositivi di I/O

Struttura del “data path”

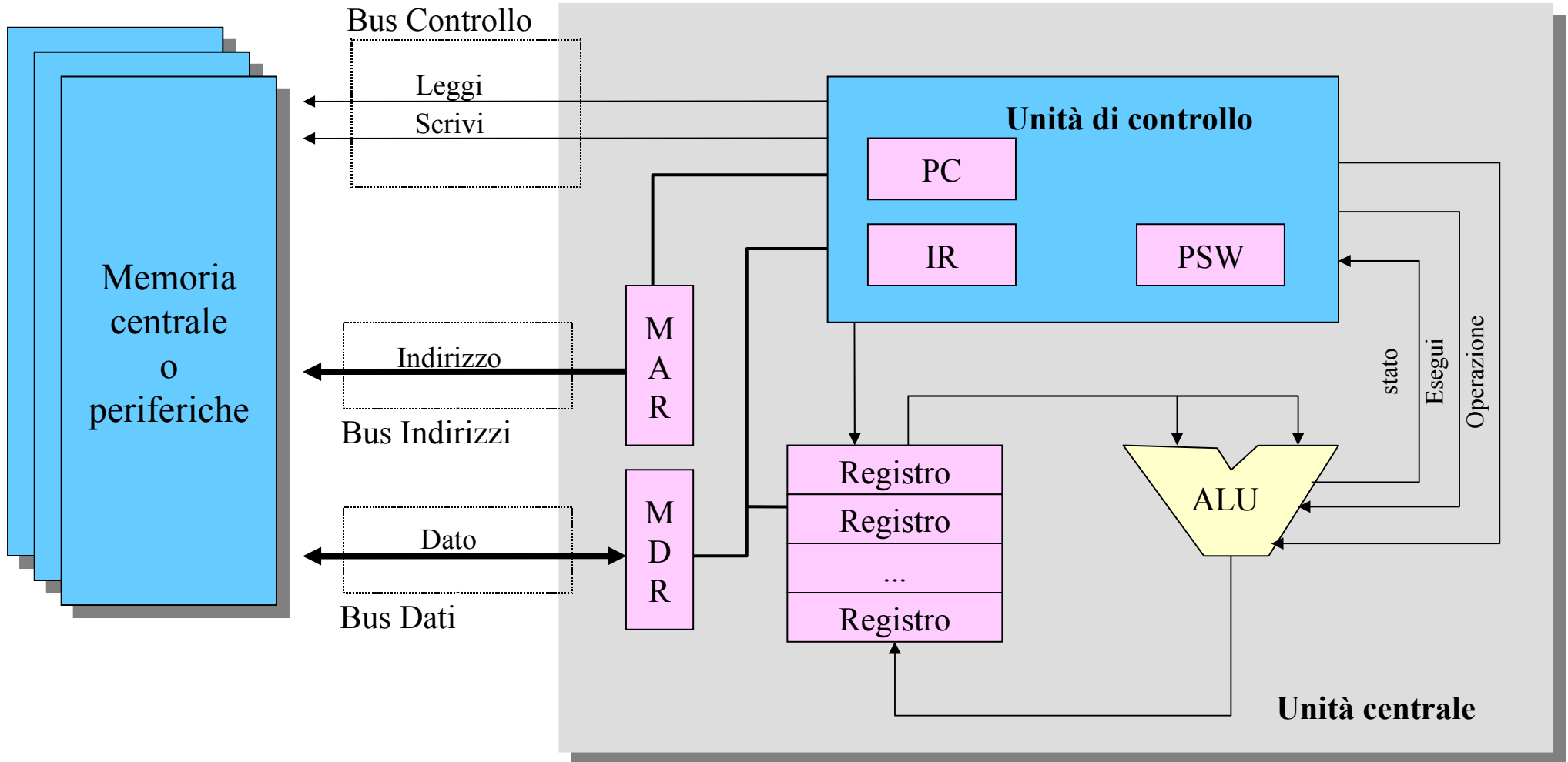


Esecuzione delle Istruzioni

■ Ciclo **Fetch-Decode-Execute**

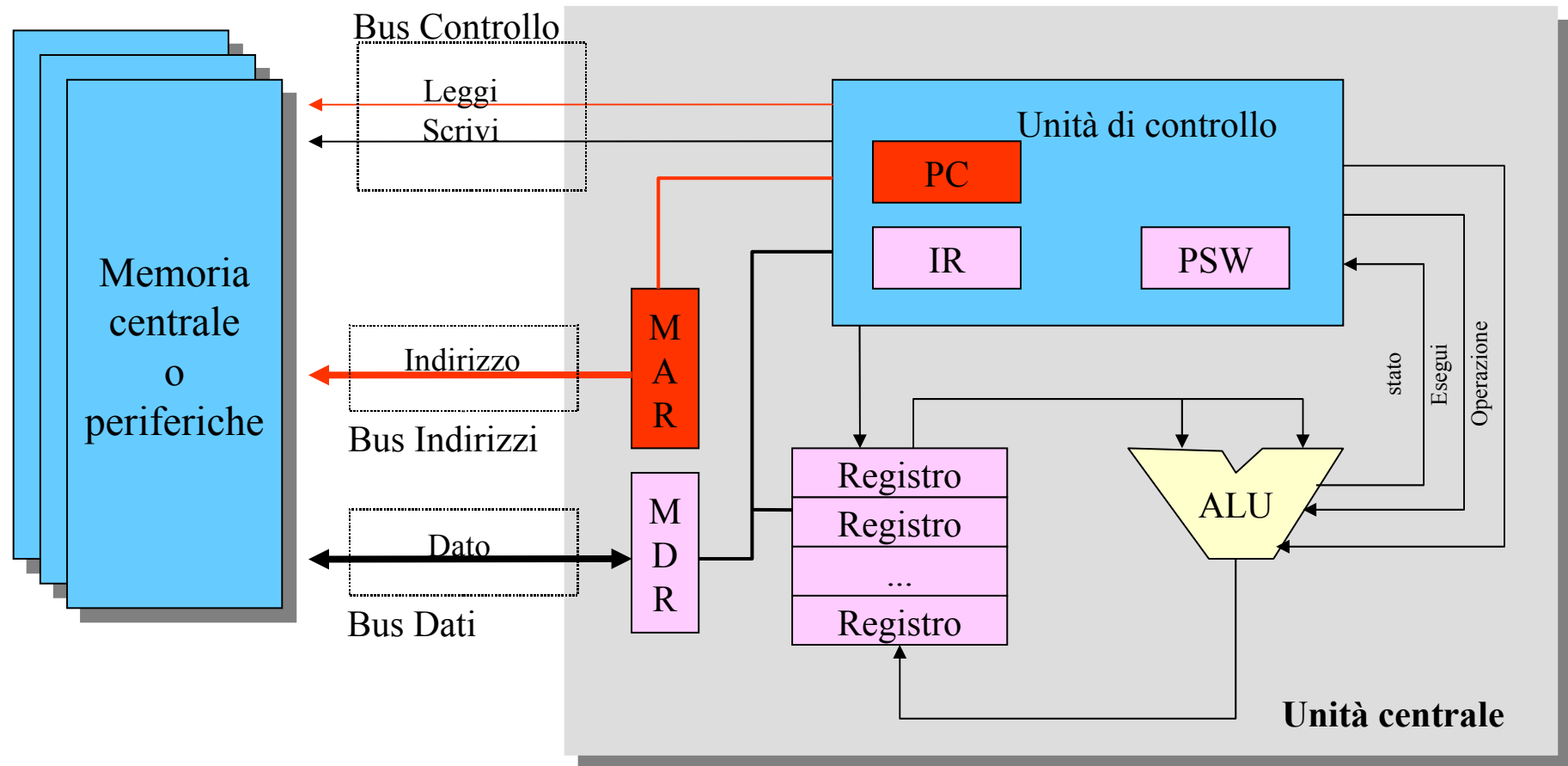
- Prendi l'istruzione corrente dalla memoria e mettila nel registro istruzioni (**IR**) [**Fetch**]
- Incrementa il program counter (**PC**) in modo che contenga l'indirizzo dell'istruzione successiva
- Determina il tipo dell'istruzione corrente [**Decodifica**]
- Se l'istruzione usa una parola in memoria determina dove si trova
- Carica la parola, se necessario, in un registro della CPU
- Esegui l'istruzione [**Execute**]
- Torna al punto 1.

Struttura Semplificata di una CPU



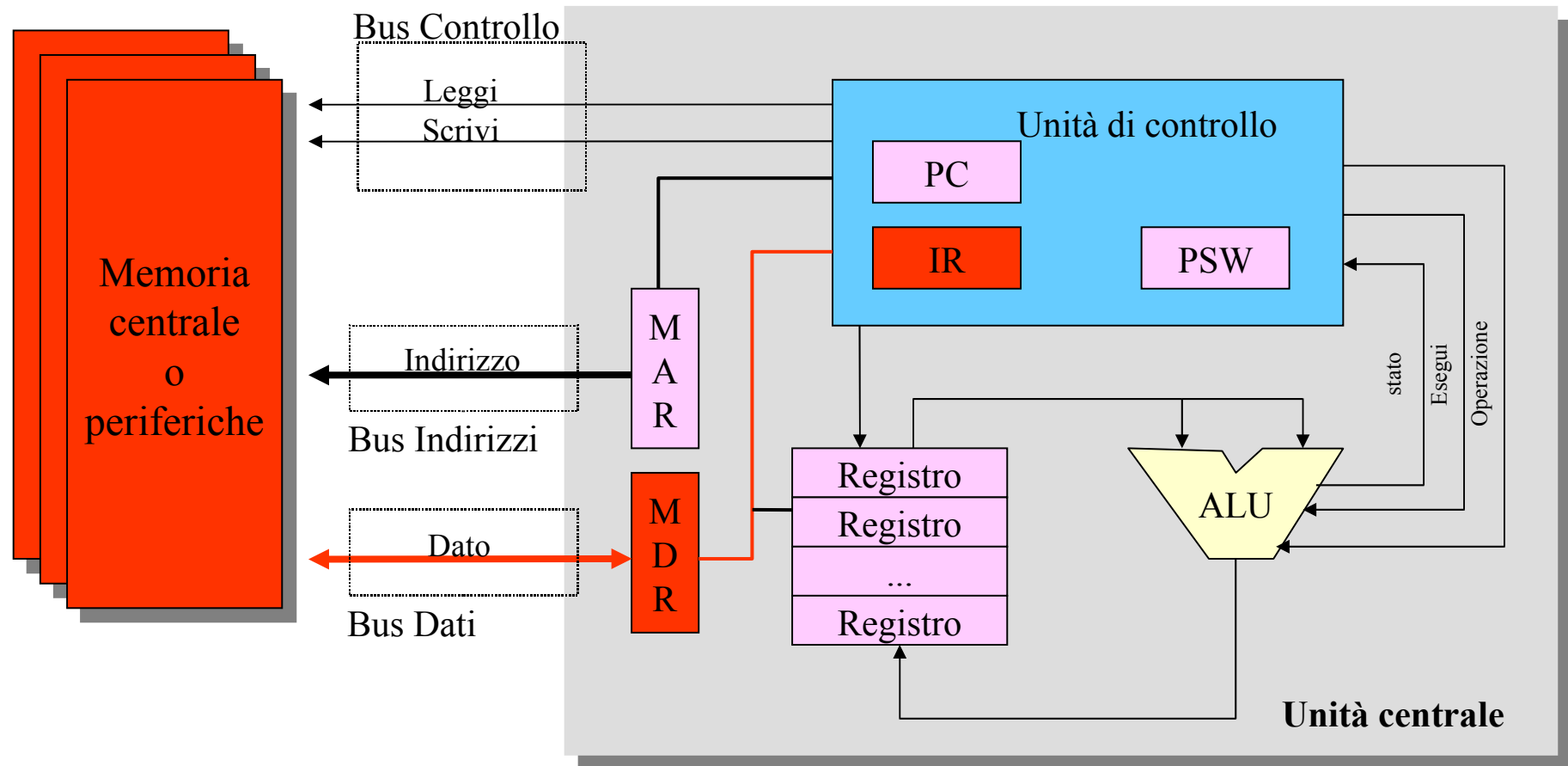
Esempio: Lettura dalla Memoria

■ Fase di Fetch (1 di 2)



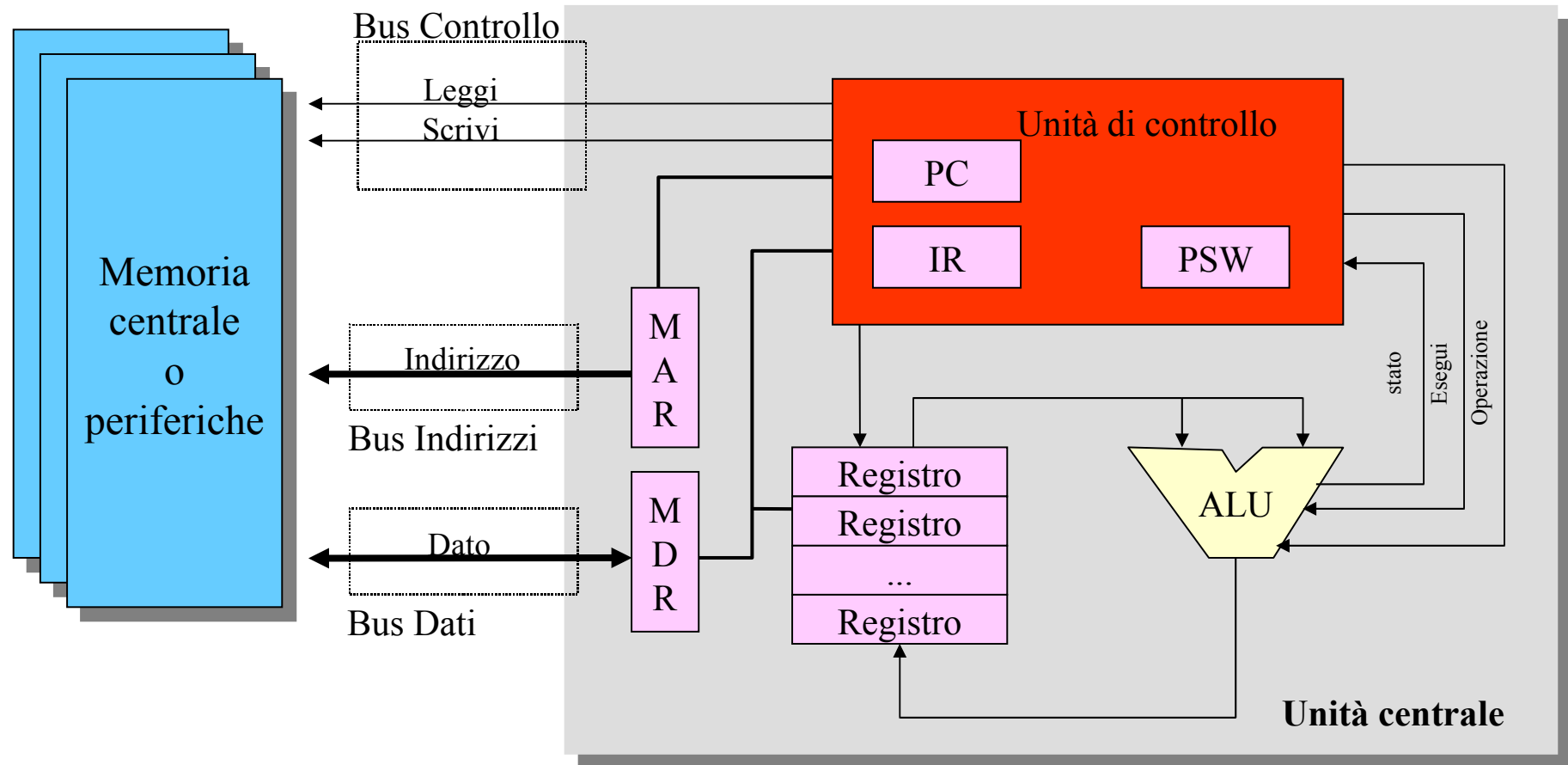
Esempio: Lettura dalla Memoria

■ Fase di Fetch (2 di 2)



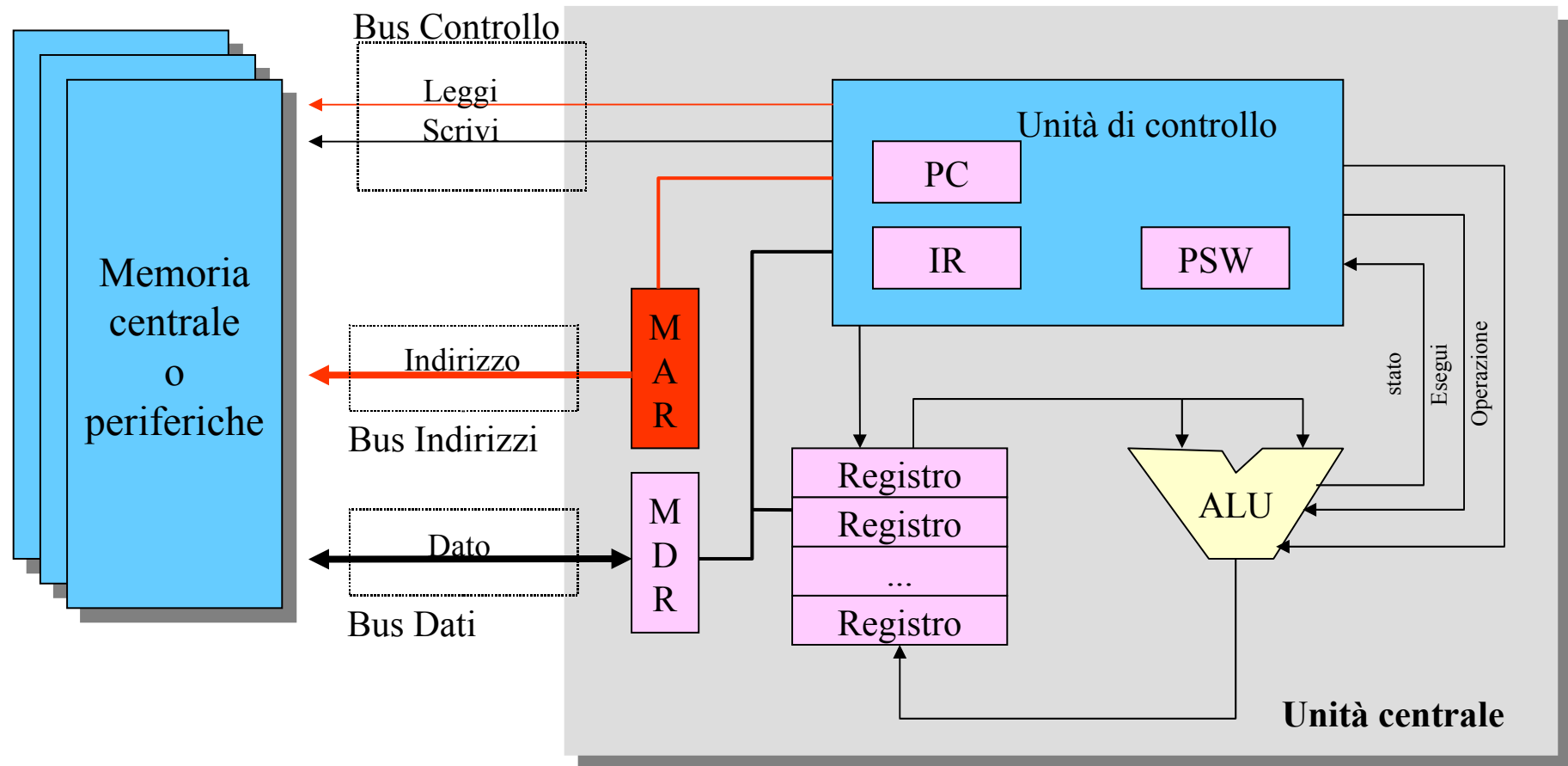
Esempio: Lettura dalla Memoria

■ Decodifica



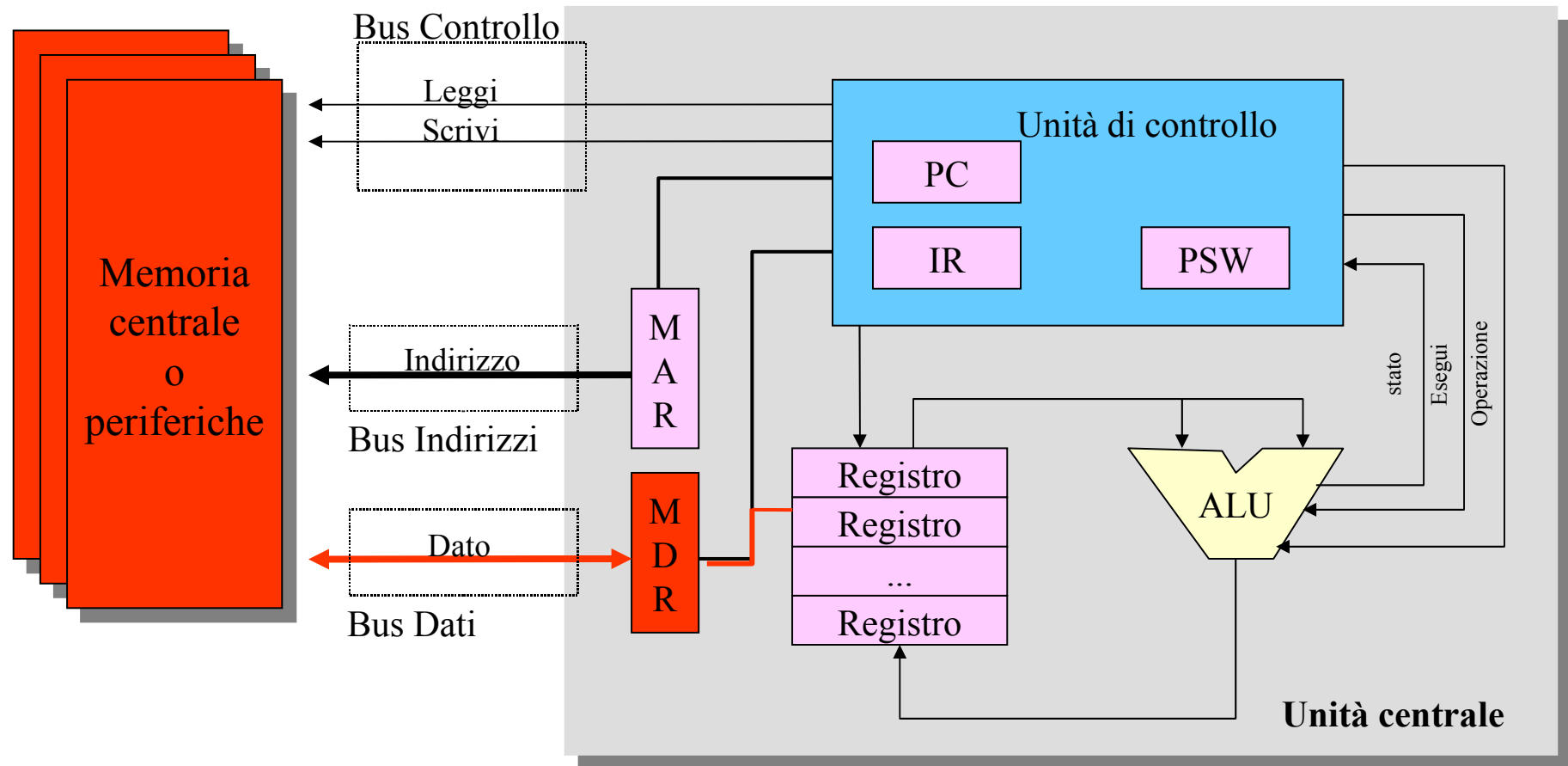
Esempio: Lettura dalla Memoria

■ Esecuzione (1 di 2)



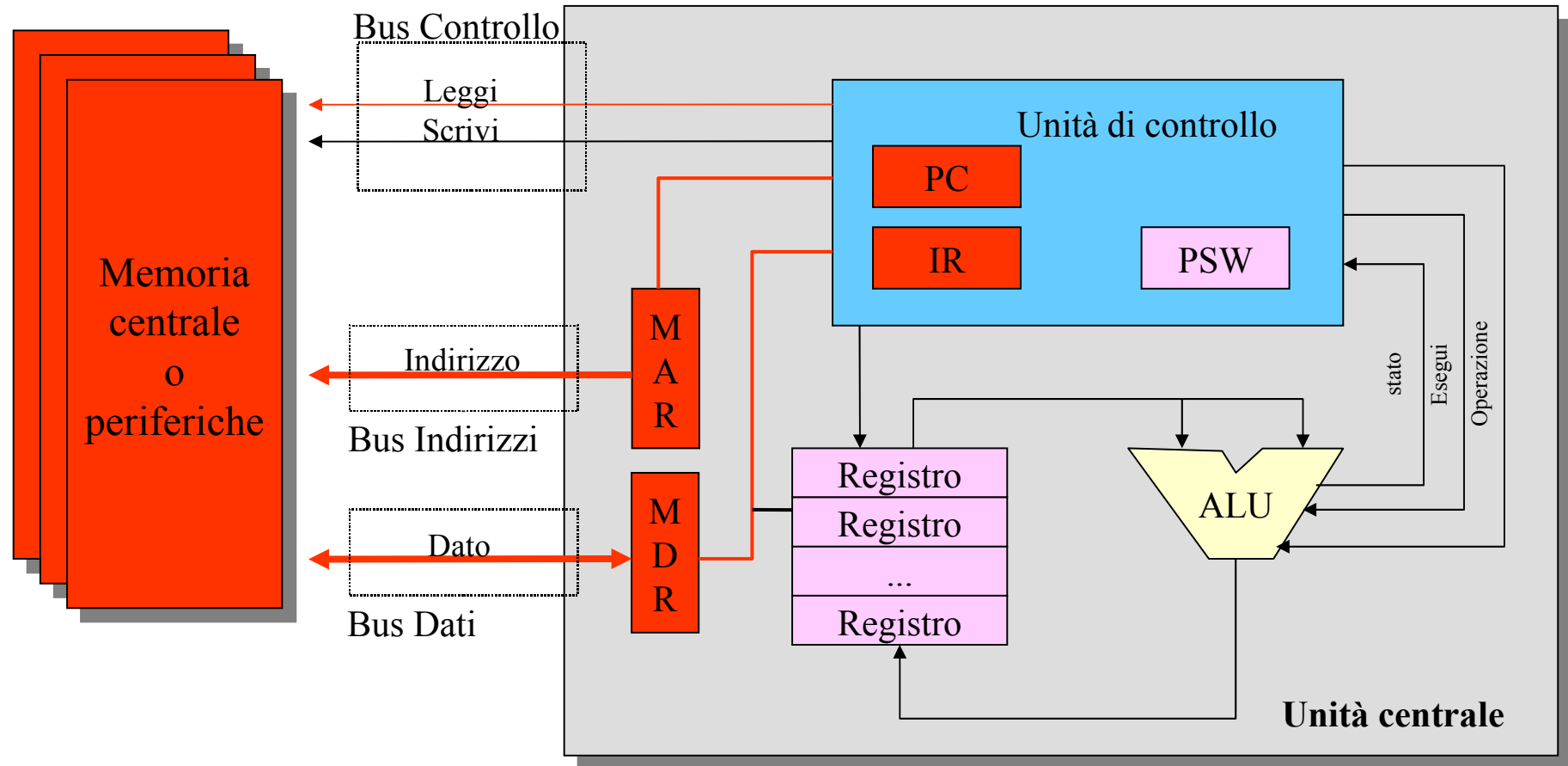
Esempio: Lettura dalla Memoria

■ Esecuzione (2 di 2)



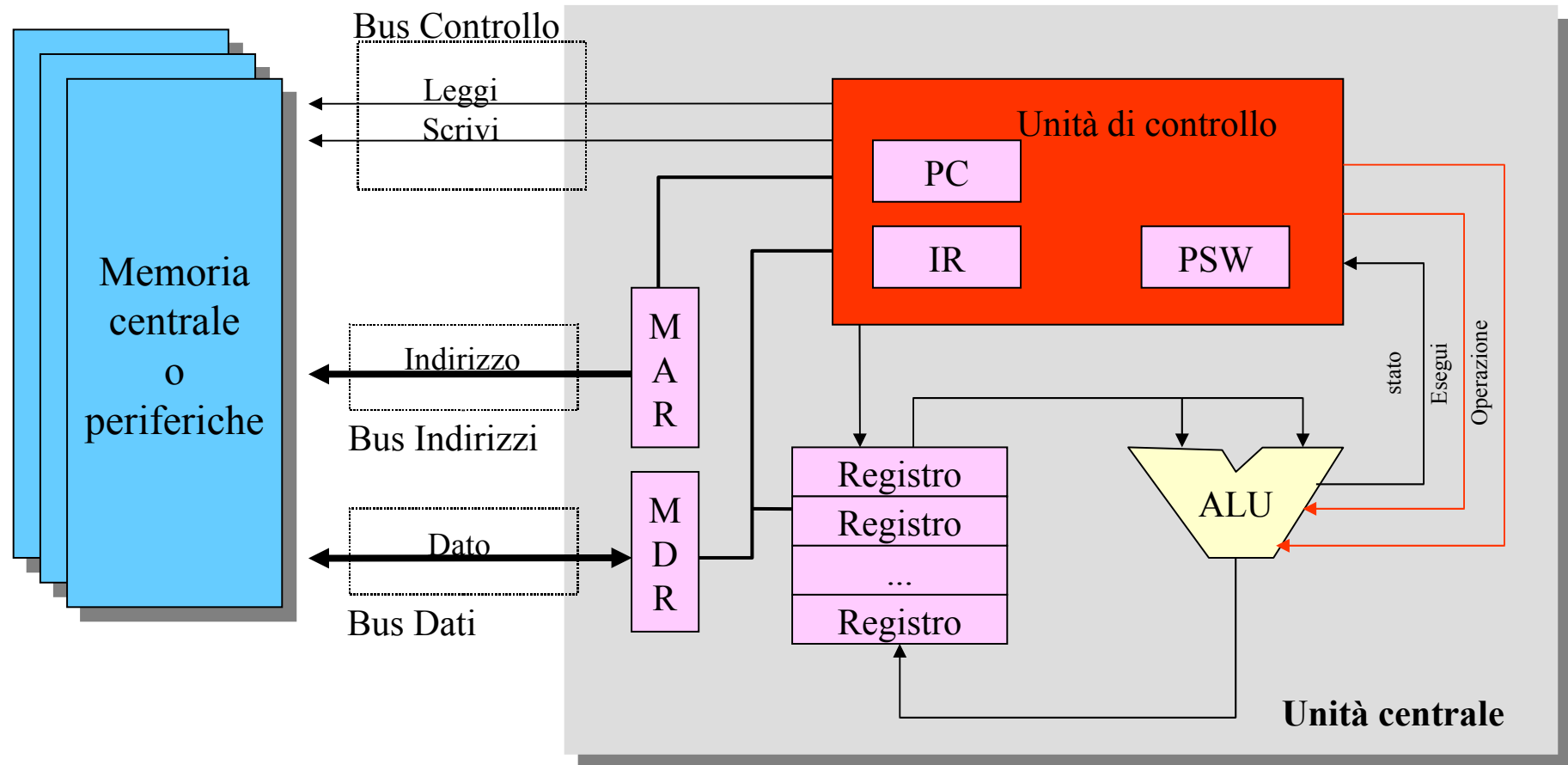
Esempio: Somma tra due registri

■ Fetch (come prima)



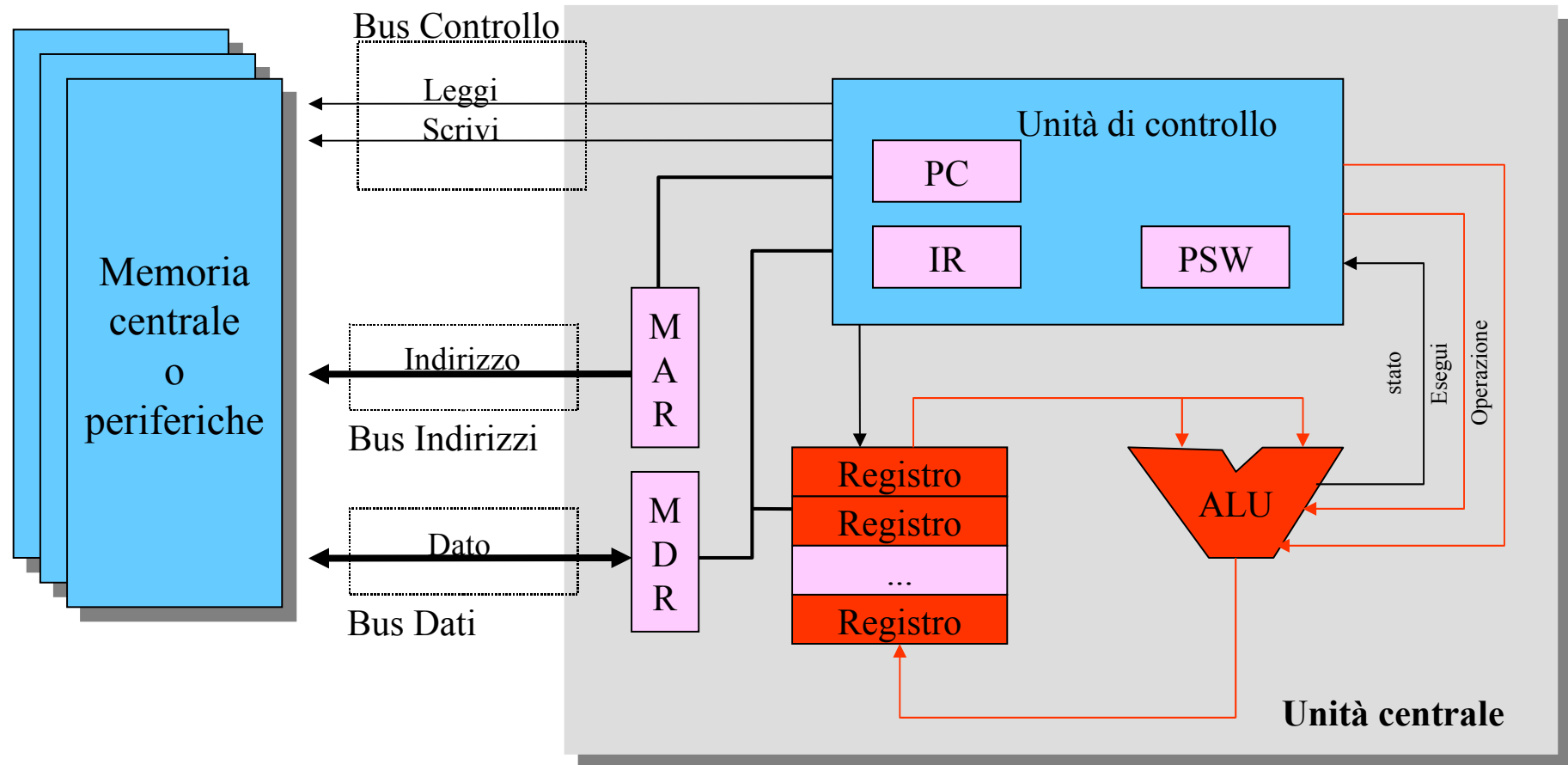
Esempio: Somma tra due registri

■ Decodifica



Esempio: Somma tra due registri

■ Esecuzione



Riepilogo: Registri di CPU

- **IR**: Usato per contenere l'istruzione in corso di esecuzione
 - Caricato in fase di fetch
 - determina le azioni svolte durante la fase di esecuzione
- **PC**: Tiene traccia dell'esecuzione del programma
 - Contiene l'indirizzo di memoria in cui è memorizzata la prossima istruzione da eseguire
- **MAR**: contiene l'indirizzo della locazione di memoria da leggere o scrivere
 - La dimensione di MAR determina l'ampiezza dello spazio di memoria fisica
 - Dalla fine degli anni '80 vengono prodotti microprocessori con bus indirizzi a 32 bit
- **MDR**: Registro attraverso il quale viene scambiata l'informazione tra la memoria e la CPU
 - Tradizionalmente la dimensione di MDR dà la misura del grado di parallelismo della macchina (8, 16, 32, 64 bit)
- **PSW**: (Program Status Word) contiene info riguardo lo stato del programma
- **R0, R1,...Rn**: Registri di uso generale

CISC

- **CISC: Complex Instruction Set Computing**
- Un repertorio di istruzioni esteso è preferibile perché:
 - ➔ Istruzioni potenti semplificano la programmazione
 - ➔ Riduce il gap tra linguaggio di macchina e linguaggio di alto livello
 - ➔ L'uso efficiente della memoria (all'epoca era costosa) era la preoccupazione principale:
 - ✓ Meglio avere codici compatti
 - ✓ Essendo (allora) la memoria di controllo molto più veloce della memoria centrale, portare funzionalità nella prima avrebbe migliorato le prestazioni della macchina

RISC

- Memorie RAM
 - Molto più veloci delle precedenti a nuclei
- Cache
 - Riducono ulteriormente i tempi di esecuzione
- Comportamento dei programmi
 - L'80% delle istruzioni eseguite corrispondeva al solo 20% del repertorio
 - Conviene investire nella riduzione dei tempi di esecuzione di quel 20%, anziché aggiungere raffinate istruzioni, quasi mai usate, ma responsabili dell'allungamento del tempo di ciclo di macchina
- Conviene costruire processori molto veloci, necessariamente con repertori semplici, e contare sull'ottimizzazione del compilatore
- **RISC**: **R**educed **I**nstruction **S**et **C**omputing

RISC - Criteri di Progettazione

- Le istruzioni devono essere semplici
 - ➔ Se l'introduzione di una operazione di macchina fa crescere del 10% il periodo di clock, allora essa deve produrre una riduzione di almeno un 10% del numero totale di cicli eseguiti
- Con memorie attuali
 - ➔ Non c'è vantaggio a spostare le funzionalità a livello di microcodice
 - ✓ Ciò ha solo l'effetto di rendere più difficoltose modifiche e cambiamenti
 - ➔ Molto meglio modificare una libreria di sistema che modificare una memoria di controllo

RISC - Criteri di Progettazione

- Tutte le istruzioni occupano lo stesso spazio di memoria (una parola)
- Ristretto numero di formati
 - ➔ L'interpretazione del codice avviene attraverso un semplice decodificatore (una rete AND-OR)
 - ➔ La codifica “ordinata” consente accorgimenti per velocizzare l'esecuzione (pipeline), difficilmente applicabili a repertori di istruzioni complesse
- La semplificazione del repertorio tende a far aumentare la dimensione del codice
 - ➔ Non è un problema, vista la tendenza alla riduzione dei costi e all'aumento della densità delle memorie
 - ➔ Dal punto di vista della velocità i guadagni che si ottengono nel semplificare le istruzioni sono superiori all'effetto negativo del maggior numero di istruzioni per programma

RISC - Criteri di Progettazione

■ Conclusioni

- ➔ Progetto di un'architettura che preveda solo operazioni tra registri (non registro/memoria o memoria/memoria) e operazioni di lettura/scrittura in memoria molto semplici con poche modalità di indirizzamento
 - ✓ Architetture Load/Store
- ➔ Il compilatore deve fare il miglior uso possibile dei registri e tenere il più possibile le variabili nei registri
 - ✓ CPU con elevato numero di registri