

# Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2022/2023

Corso di Laurea in Matematica

## Introduzione al SO Architettura di un elaboratore

Annalisa Massini

Dipartimento di Informatica  
Sapienza Università di Roma

# Informazioni Generali sul Corso

- **Orario**

- lunedì 15:00-18:00

- **Testi di riferimento**

- W. Stallings, *Operating Systems, Internals and Design Principles*, disponibile la settima edizione al link:  
[https://dinus.ac.id/repository/docs/ajar/Operating\\_System.pdf](https://dinus.ac.id/repository/docs/ajar/Operating_System.pdf)
- F. C. A. Johnson, *Pro Bash Programming (Scripting the GNU/Linux Shell)*, Apress
- slide e dispense che verranno messe a disposizione sulla pagina del corso

- **Modalità esame** - scritto e prova di laboratorio

# Argomenti della lezione

- 1 **Introduzione**
  - I sistemi operativi
- 2 **Architettura di un elaboratore**
  - Descrizione generale
  - I registri
  - Le istruzioni
  - Le interruzioni
  - La memoria
  - Input/Output

## Introduzione

# I sistemi operativi

# Sistema Operativo

- Un sistema operativo è un programma (più precisamente una collezione di programmi) che:
  - consente di eseguire le operazioni fondamentali della macchina
  - inizializza il sistema e assicura il corretto funzionamento della macchina
  - permette di eseguire qualsiasi altro programma o pacchetto software

# Sistema Operativo

Il sistema operativo:

- sfrutta le risorse hardware di un sistema computerizzato
  - uno o più processori
  - memoria primaria (RAM)
  - memoria secondaria (dischi)
  - dispositivi di input/output
  
- e fornisce un insieme di servizi agli utenti
  - in particolare, offre un ambiente di esecuzione **facilitato** per le applicazioni utente



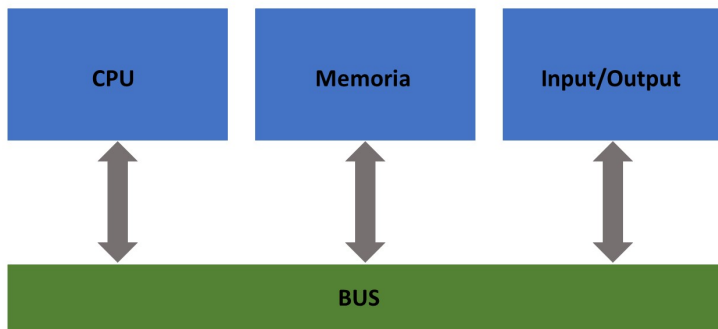




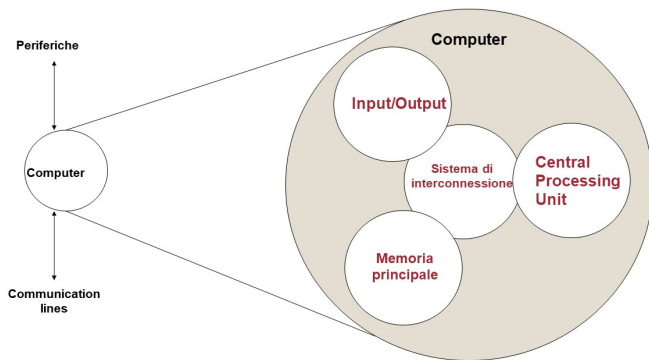
# Definizione di elaboratore

- Un elaboratore è un sistema complesso che contiene milioni di componenti elettronici di base
- Le funzioni base di un elaboratore sono:
  - Elaborazione di dati (o informazioni) - **Data processing**
    - le informazioni possono avere una grande varietà di formati
  - Memorizzazione di dati (o informazioni) - **Data storage**
    - in particolare occorre memorizzare almeno le informazioni che vengono elaborate in un dato momento
  - Spostamento di dati (o informazioni) - **Data movement**
    - informazioni (dati) devono poter essere trasferite sia all'interno dell'elaboratore che verso il mondo esterno
  - Attività di controllo - **Control**
    - per supervisionare (**controllare**) l'esecuzione delle tre funzioni precedenti

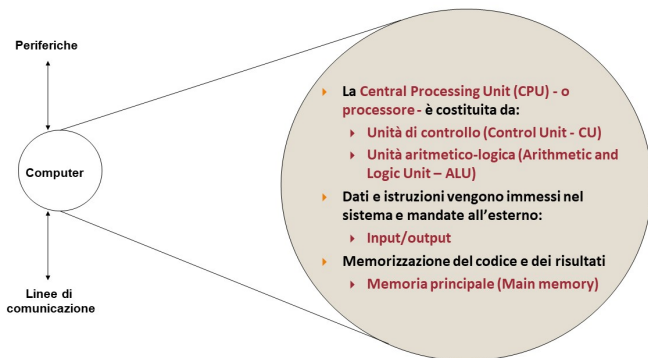
# Architettura di Von Neumann



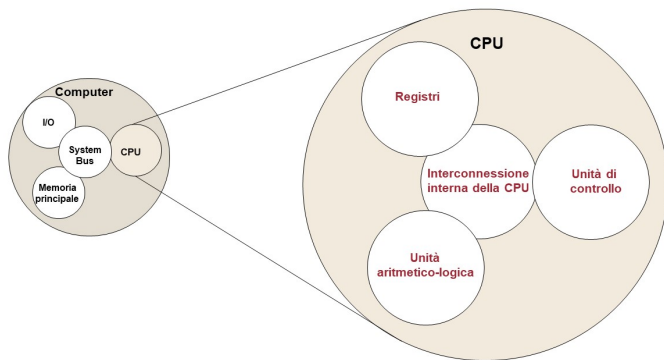
# Struttura ad alto livello



# Struttura ad alto livello



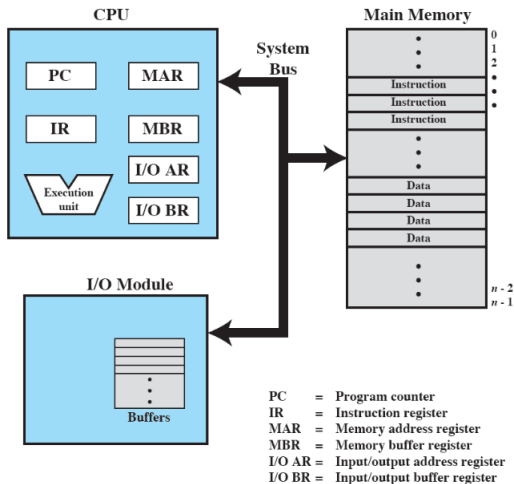
# Struttura ad alto livello



# Nozioni di Base: Parti Principali

- Processore
  - il cervello del computer: si occupa di tutte le computazioni
- Memoria Principale
  - volatile: se si spegne il computer, se ne perde il contenuto
  - talvolta chiamata memoria reale o primaria
- Moduli di input/output
  - dispositivi di memoria secondaria (dischi...), non volatile
  - dispositivi per la comunicazione (schede di rete...)
  - altri dispositivi: tastiera, monitor, stampante, mouse, ...
- Bus di sistema
  - mezzo per far comunicare tra loro le parti interne del computer: processori, memoria principale, e moduli di input/output

# Componenti di un Computer



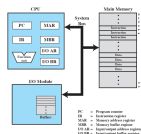
## Architettura di un elaboratore

# I registri



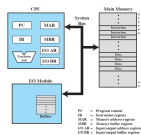
# Registri del Processore

- Registri di controllo e di stato
  - usati dal processore per controllare l'uso del processore stesso
  - usati da funzioni privilegiate del SO per controllare l'esecuzione dei programmi
- Registri *interni*
  - usati dal processore per l'esecuzione delle istruzioni
- Registri visibili dall'utente
  - piccolo insieme di registri (veloci e costosi) usati per evitare l'accesso alla memoria principale
  - usati da chi programma in assembler o dai compilatori di linguaggi *non* interpretati



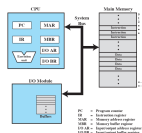
# Registri di Controllo e Stato

- Contatore di Programma (Program Counter, o PC)
  - contiene l'indirizzo di un'istruzione da prelevare dalla memoria
- Registro di Istruzione (Instruction Register, o IR)
  - contiene l'istruzione prelevata più di recente
- Stato di Programma (Program Status Word, o PSW)
  - contiene le informazioni di stato, ad es: interrupt disabilitati
- Codici di condizione (o flag)
  - singoli bit settati dal processore come risultato di operazioni
  - esempi: risultato positivo, negativo, zero, overflow, ...



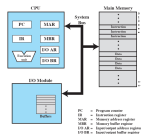
# Registri di Controllo e Stato

- Vengono usualmente letti/modificati in modo *implicito* da opportune istruzioni assembler
  - esempio: una jump modifica il PC
- Nel Pentium sono considerati registri di controllo anche quelli per la gestione della memoria
  - ad esempio, i registri cr0 ... cr4 gestiscono le tabelle delle pagine
  - un bit di cr0 abilita la paginazione tout-court



# Registri Interni

- Registro dell'indirizzo di memoria - MAR (Memory Address Register)
  - contiene l'indirizzo della prossima operazione di lettura/scrittura
- Registro di memoria temporanea - MBR (Memory Buffer Register)
  - contiene i dati da scrivere in memoria, o fornisce lo spazio dove scrivere i dati letti dalla memoria
- Registro dell'indirizzo di input/output - I/O address register
- Registro di memoria temporanea per l'input/output - I/O buffer register



# Registri Visibili dall'Utente

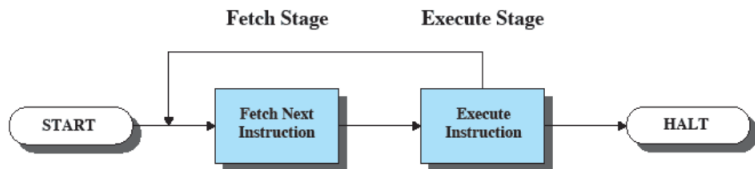
- Gli unici che possono essere usati *direttamente* (con il loro nome) quando si programma in linguaggio macchina (o assembler), ad es.:
  - `mov $5, %eax` (Pentium con sintassi AT&T)
  - `li $t1, 5` (MIPS, ad es. R3000A nella PlayStation)
- Possono contenere dati o indirizzi
- Nel caso contengano indirizzi, possono essere
  - puntatori diretti
  - registri-indice: per ottenere l'indirizzo effettivo, occorre aggiungere il loro contenuto ad un indirizzo base
  - puntatori a segmento: se la memoria è divisa in segmenti, contengono l'indirizzo di inizio di un segmento

## Architettura di un elaboratore

# Le istruzioni

# Esecuzione di un'istruzione

- Due passi
  - Il processore legge (preleva, fase di fetch) istruzioni dalla memoria (principale)
  - Il processore esegue ogni istruzione prelevata



# Prelievo ed Esecuzione di un'istruzione

- Il processore preleva l'istruzione dalla memoria principale
- Dopo ogni prelievo il PC è incrementato
- Il PC mantiene l'indirizzo della successiva istruzione da prelevare
- Se l'istruzione è una jump, il PC verrà ulteriormente modificato dall'istruzione stessa caricando l'indirizzo di destinazione del salto



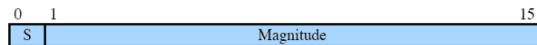
# Registro Istruzione

- L'istruzione prelevata viene caricata nell'IR
- Categorie di istruzioni
  - Scambio dati tra processore e memoria
  - Scambio dati tra processore e input/output
  - Manipolazione di dati
    - include operazioni aritmetiche
    - solitamente solo con registri, ma in alcuni processori anche direttamente in RAM
  - Controllo
    - modifica del PC tramite salti condizionati o non
  - Operazioni riservate
    - disabilitazione interrupt
    - disabilitazione cache

# Esempio su una macchina ipotetica



(a) Instruction format



(b) Integer format

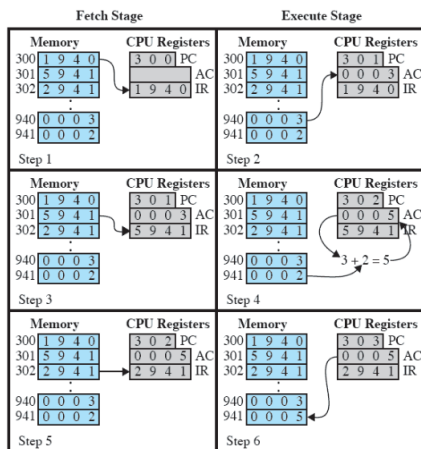
Program counter (PC) = Address of instruction  
 Instruction register (IR) = Instruction being executed  
 Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from memory  
 0010 = Store AC to memory  
 0101 = Add to AC from memory

(d) Partial list of opcodes

## Esempio di esecuzione di un programma



## Architettura di un elaboratore

# Le interruzioni

# Interruzioni

- Le **interruzioni** interrompono la normale esecuzione sequenziale del processore
  - come conseguenza, viene eseguito del **software di sistema**, che tipicamente non è stato scritto dall'utente che sta eseguendo un certo programma
- Le **cause di un'interruzione** sono molteplici e danno luogo a diverse **classi** di interruzioni:
  - da I/O
  - da fallimento hardware
  - da programma
  - da timer

# Classi di Interruzioni *Asincrone*

- **Interruzioni da input/output**
  - generate dal controllore di un dispositivo di input/output
  - per la maggior parte, i dispositivi di input/output sono più lenti del processore
  - quindi, il processore deve mettersi in pausa per aspettare che il dispositivo completi l'operazione corrente
  - segnalano il completamento o l'errore di un'operazione di I/O
- **Interruzioni da fallimento HW**
  - Improvvisa mancanza di potenza (power failure)
  - errore di parità nella memoria

## Classi di Interruzioni *Asincrone*

- **Interruzioni da comunicazione tra CPU**
  - per sistemi dove ce n'è più d'una
- **Interruzioni da timer**
  - generate da un timer interno al processore
  - permettono al sistema operativo di eseguire alcune operazioni ad intervalli regolari
- Per processori Intel, gli *interrupt* sono solo questi

# Classi di Interruzioni *Sincrone*

- **Interruzioni di programma** - causate principalmente da:
  - overflow
  - divisione per 0
  - debugging: single step o breakpoint
  - tentativo di esecuzione di un'istruzione macchina errata
    - opcode illegale, oppure operando non allineato
  - chiamata a *system call* (molto spesso)
  - ...



## Classi di Interruzioni *Sincrone*

- Interruzioni di programma, causate principalmente da:
  - ...
  - riferimento ad indirizzo di memoria fuori dallo spazio disponibile al programma
  - riferimento ad indirizzo di memoria momentaneamente non disponibile
    - memoria virtuale: ci ritorneremo
  - tentativo di serializzare 2 eccezioni non serializzabili (raro)

# Interruzioni ed istruzione di ritorno

- Per le interruzioni asincrone, una volta che l'**handler** è terminato, si riprende dall'istruzione subito successiva a quella dove si è verificata l'interruzione
- In realtà, potrebbe succedere che ci sia un *process switch*, ma di questo parleremo più avanti
- Comunque, quando la computazione ritornerà al processo interrotto, si ricomincia dall'istruzione successiva

# Interruzioni ed istruzione di ritorno

- Con le interruzioni sincrone, non è detto che si riprenda con l'istruzione successiva
  - **faults**: errore correggibile, viene *rieseguita la stessa istruzione*
    - es.: page fault
  - **aborts**: errore non correggibile, computazione terminata
    - es.: segmentation fault
  - **traps** e **system calls**: si continua dall'istruzione successiva
    - es. per traps: debugging

# Fase di interruzione

- Ad ogni ciclo fetch-execute, viene anche controllato se c'è stata un'interruzione (o una exception)
- Se è così il programma viene *sospeso* e viene eseguita una funzione che gestisce l'interruzione (interrupt-handler routine)

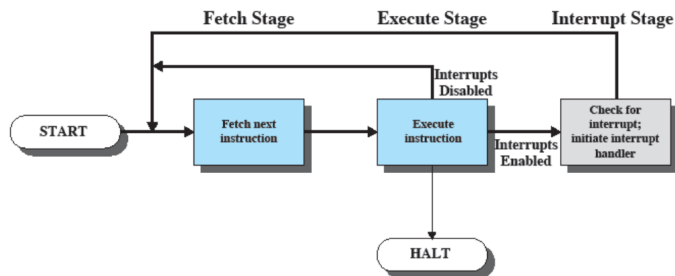
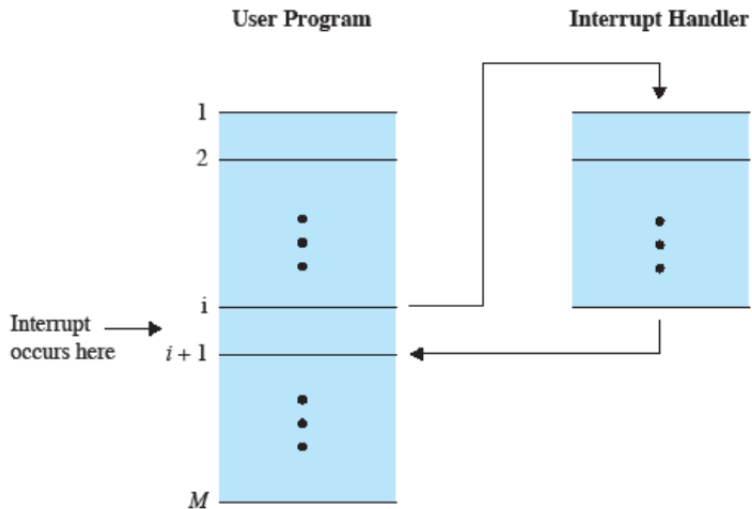


Figure 1.7 Instruction Cycle with Interrupts

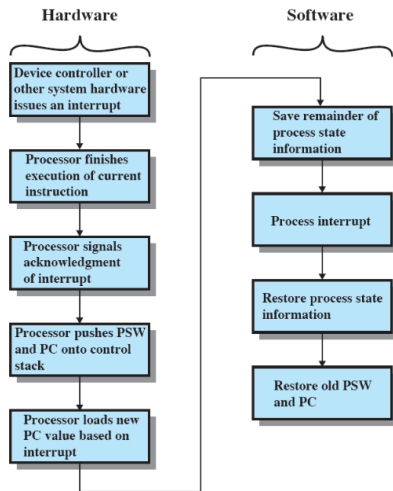
# Interruzioni: trasferimento del controllo

- L'**interrupt handler** è una funzione particolare: nel programma utente non era prevista
- Sistema operativo e hardware collaborano per salvare le informazioni (almeno registro di stato e Program Counter) e settare il Program Counter al nuovo valore
  - normalmente, questo lo fa il *programmatore* (se scrive direttamente in assembler) o il *compilatore*
  - corrisponde a una chiamata a funzione, ma non è stata scritta dal programmatore (niente di strano, esistono le librerie...)
  - Infatti nel punto in cui avviene questa *chiamata*, il programma utente non prevedeva affatto di effettuare la chiamata all'interrupt handler!
  - ma comunque si sa che le *interruzioni vere* - **eccezioni** - possono essere sollevate

## Interruzioni: trasferimento del controllo

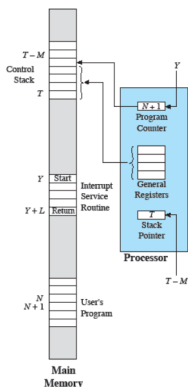


# Interruzioni: trasferimento del controllo



# Interruzioni: modifiche a memoria e registri

Il processore sta eseguendo l'istruzione all'indirizzo  $N$  quando arriva un'interruzione, da gestire con l'handler all'indirizzo  $Y$

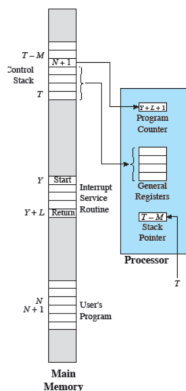


(a) Interrupt occurs after instruction at location  $N$



# Interruzioni: modifiche a memoria e registri

Quando l'handler è completato, si va all'indirizzo  $N + 1$ , **ma** se l'interruzione era una *fault* correggibile (es. un page fault), si va all'indirizzo  $N$



(b) Return from interrupt

## Interruzioni disabilitate

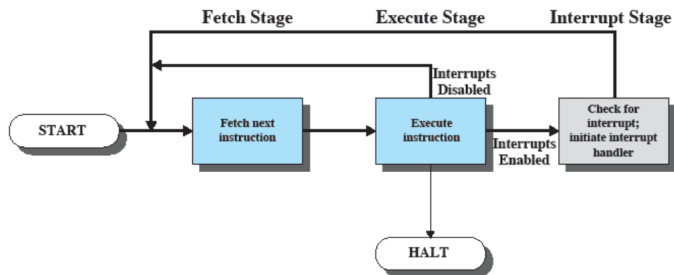
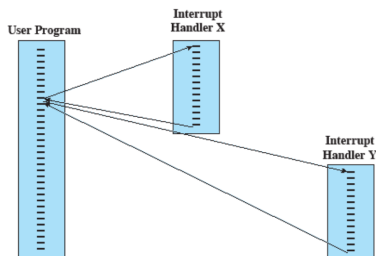
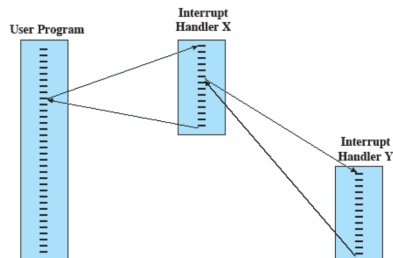


Figure 1.7 Instruction Cycle with Interrupts

# Interruzioni: sequenziali ed annidate

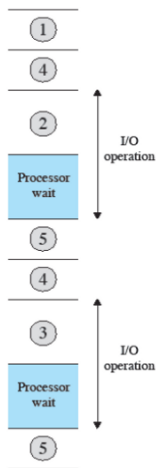


(a) Sequential interrupt processing



(b) Nested interrupt processing

# Multiprogrammazione

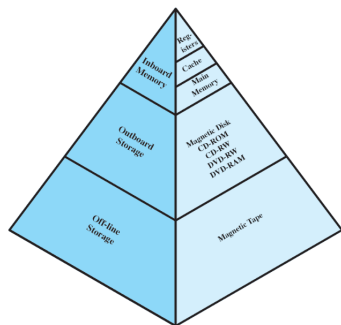


- Un processore deve eseguire più programmi contemporaneamente
- La sequenza con cui i programmi sono eseguiti dipende dalla loro priorità e dal fatto che siano o meno in attesa di input/output
- Alla fine della gestione di un'interruzione, il controllo potrebbe non tornare al programma che era in esecuzione al momento dell'interruzione

## Architettura di un elaboratore

# La memoria

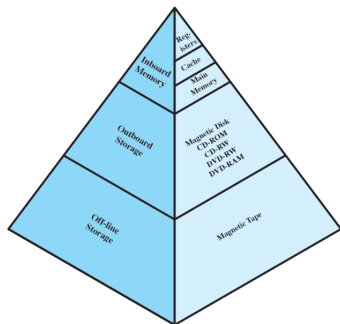
# Gerarchia di memoria



## Dall'alto verso il basso

- Diminuisce la velocità di accesso
- Diminuisce il costo al bit
- Aumenta la capacità
- Diminuisce la frequenza di accesso alla memoria da parte del processore

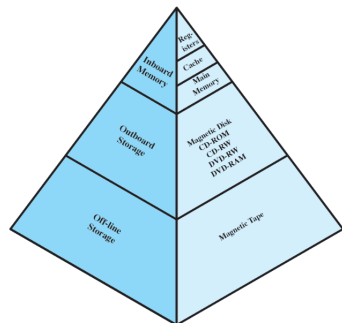
# Gerarchia della memoria



## La memoria secondaria:

- corrisponde alla memoria *outboard* e all'offline storage
- è una memoria *ausiliaria* ed *esterna*
- è non volatile: se si spegne il computer, il contenuto rimane
- è usata per memorizzare i file contenenti programmi o dati

# Gerarchia della memoria

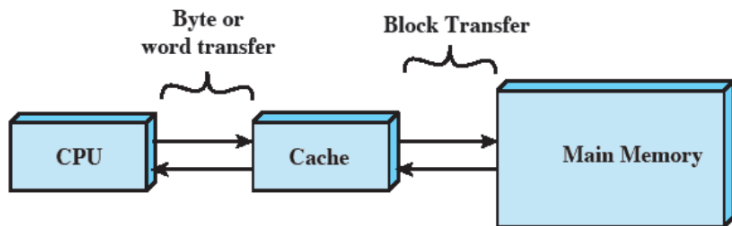


## La memoria principale:

- anche nella memoria *inboard* ci sono importanti differenze di velocità
- la velocità del processore è maggiore della velocità di accesso alla memoria principale
- quindi, per evitare tempi di attesa eccessivi, tutti i computer hanno una *memoria cache*
- La cache è una memoria piccola e veloce, che sfrutta il principio di località



# Cache e Memoria Principale



# Cache: Nozioni di Base

- Contiene copie di porzioni della memoria principale
- Il processore prima controlla se un dato è nella cache
- Se non c'è, il corrispondente blocco di memoria principale viene caricato nella cache
- Siccome vale la *località dei riferimenti*, è probabile che il dato appena caricato serva ancora nell'immediato futuro
  - in generale, i futuri riferimenti in memoria ricadranno probabilmente nel blocco appena caricato
- Gestione totalmente demandata all'hardware
  - il programmatore non “vede” la cache, neanche se usa l'assembler
  - quindi, non la “vede” neanche un compilatore
  - e neanche il sistema operativo, che però la “imita” spesso

## Cache vs. Memoria Principale

$$\frac{2^n}{K} \gg C; |Tag| = \log_2 \frac{2^n}{K}; \text{tipicamente, } K = 4$$

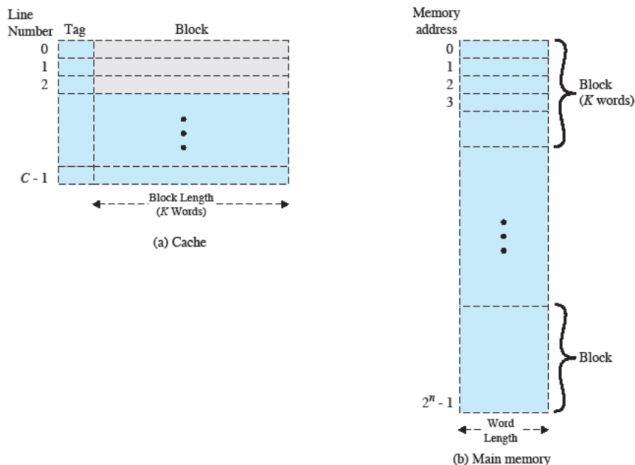
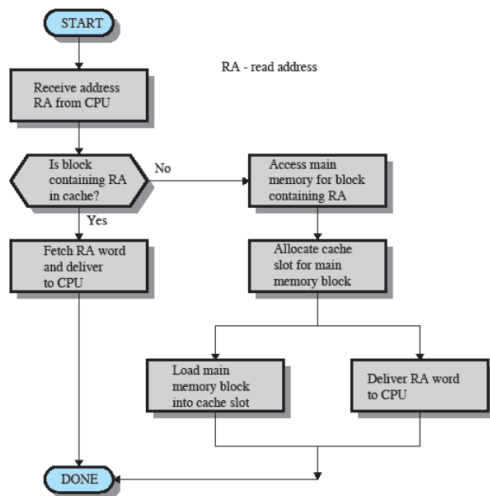


Figure 1.17 Cache/Main-Memory Structure

## Lettura dalla Cache



# Cache: Nozioni di Base

- Capacità della cache
  - cache anche piccole hanno un grande impatto sulle performance
- Misura dei blocchi
  - i dati scambiati tra memoria e cache sono in quantità multiple di un blocco
  - incrementare la misura del blocco aumenta il numero di accessi riusciti
  - ma incrementarla troppo è controproducente: saranno di più anche i dati che vengono rimossi
  - il che abbassa la probabilità di accesso riuscito

# Cache: Nozioni di Base

- Funzione di mappatura
  - determina la locazione della cache nella quale andrà messo il blocco proveniente dalla memoria
- Algoritmo di rimpiazzamento
  - sceglie il blocco da rimpiazzare
  - algoritmo Least-Recently-Used (LRU): si rimpiazza il blocco usato meno di recente
- Politica di scrittura
  - determina quando occorre scrivere in memoria
  - può accadere ogni volta che un blocco viene modificato (*write-through*)
  - può accadere quando il blocco è rimpiazzato (*write-back*)
  - occorre minimizzare le operazioni di scrittura
  - questo vuol dire che la memoria può trovarsi in uno stato “obsoleto”, ovvero non in linea con il contenuto della cache

## Architettura di un elaboratore

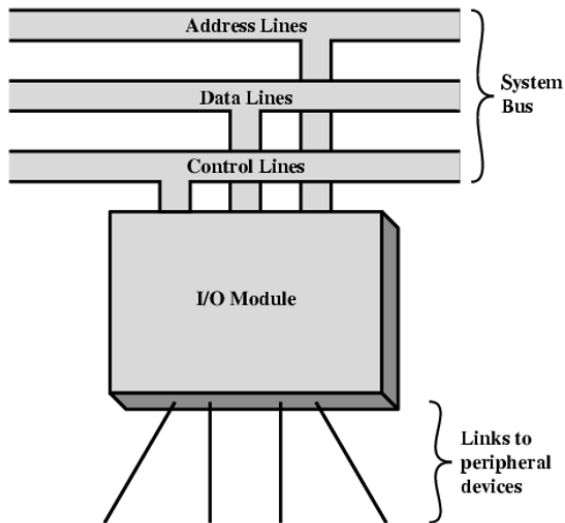
# Input/Output

# Generalità sui meccanismi di I/O

- Grande varietà di dispositivi per I/O (periferiche)
  - gestiscono diverse quantità di dati
  - lavorano a diverse velocità
  - gestiscono dati in formati diversi
- I dispositivi di I/O sono più lenti della CPU e della RAM
- Necessità di **moduli di I/O** per liberare la CPU delegando al modulo di I/O il controllo del trasferimento dei dati tra memoria e periferiche



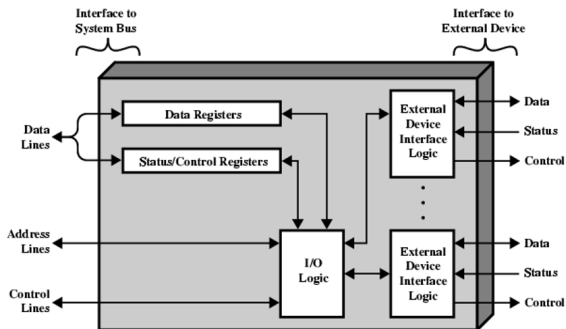
# Moduli di I/O



# Moduli di I/O

- Un modulo di Input/Output fornisce:
  - un'interfaccia tra CPU e memoria
  - un'interfaccia con una o più periferiche
- Azioni del modulo di I/O
  - il CPU controlla lo stato del modulo di I/O
  - il modulo di I/O restituisce il suo stato
  - se il modulo di I/O è pronto, la CPU richiede il trasferimento di dati
  - il modulo richiede i dati alla periferica
  - il modulo trasferisce i dati alla CPU

## Moduli di I/O



# Moduli di I/O

- Nel tempo sono state adottate soluzioni sempre più efficienti per eseguire il trasferimento di dati tra memoria e periferiche
  - Lo scopo principale è liberare la CPU dal compito di eseguire il trasferimento dati
  - Il motivo è la grande differenza di velocità della CPU rispetto alle periferiche
- Tipi di realizzazione di I/O
  - I/O programmato - metodo più vecchio e più lento perchè gestito dalla CPU che lavora secondo i tempi delle periferiche
  - I/O da interruzioni - metodo più efficiente del precedente il cui la CPU è coinvolta quando porzioni di dati sono pronti per il trasferimento ed è richiesta tramite interruzione
  - Accesso diretto alla memoria - Direct Memory Access (DMA) - metodo più efficiente in cui la CPU demanda la gestione del trasferimento dati ad un modulo dedicato, cioè il controllore di DMA