

# Parte V

## Indice

---

- ◆ Architettura di un sistema di elaborazione
  - la CPU
  - la memoria
  - le memorie di massa
  - i dispositivi di I/O
- ◆ Funzionamento di un sistema di elaborazione
  - concetto di programma
  - cenni sui linguaggi di programmazione
  - sistemi operativi
  - compilatore e linker
- ◆ Esercizi

# Sistema di elaborazione

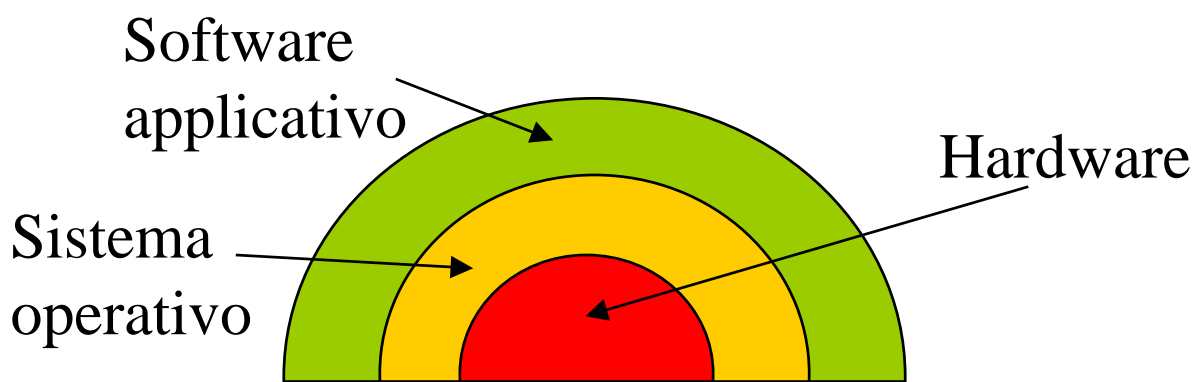
---

- ◆ Il termine *sistema di elaborazione* (SE) indica solitamente un dispositivo in grado di eseguire una serie di operazioni a seguito di comandi impartiti dall'utente
- ◆ Un sistema di elaborazione è dotato di una serie di dispositivi che gli permettono di comunicare con “il mondo esterno”, ricevendo dati e comandi e trasmettendo all'esterno i risultati delle sue elaborazioni

# Sistema di elaborazione

---

- ◆ Nel senso più generale del termine un sistema di elaborazione è caratterizzato da due parti:
  - l'hardware che è l'insieme delle parti elettroniche e meccaniche del sistema
  - il software che è l'insieme dei programmi che possono essere eseguiti dal sistema
- ◆ Il seguente diagramma (“struttura a cipolla”) schematizza i livelli di un sistema di elaborazione:



# Classificazione dei sistemi di elaborazione

---

- ◆ I sistemi di elaborazione sono usualmente classificati nelle seguenti categorie:
  - *super computer*: sistemi multiutente e multiprocessore, utilizzati per il calcolo scientifico intensivo (es. previsioni meteorologiche)  
costo: > 10 MLD
  - *mainframe*: sistemi multiutente di fascia alta (> 100 utenti), potenza di calcolo elevata, costo: 1-10 MLD
  - *mini computer e super-mini*: sistemi multiutente di fascia media, media potenza, costo: 100-1000 MIL
  - *micro computer e supermicro*: sistemi multiutente di fascia bassa, costo: 10-100 MIL

# Classificazione dei sistemi di elaborazione

---

## ◆ Classificazione (*segue*):

- *workstation*: calcolatori in genere mono-utente dotati di dispositivi e software di qualità superiore, costo: 10-100 MIL
- *personal computer*: calcolatori personali di piccole dimensioni, economici e mono-utente, costo: 1-10 MIL
- *laptop e notebook*: personal computer di ridotte dimensioni al fine di renderli trasportabili
- *palmtop*: personal computer di ridotta potenza utilizzati come agende elettroniche

# Classificazione dei sistemi di elaborazione

---

## ◆ Classificazione (*segue*):

- *home computer*: calcolatori di bassa potenza e molto essenziali (il video è un televisore, es. Commodore 64)
- *console*: evoluzione dei precedenti, dedicati esclusivamente all'uso ludico

## ◆ Tendenza attuale:

- Home: scomparsi, soppiantati dalle console
- PC e WS: sempre più simili
- sistemi multiutente: in declino a favore dei precedenti, le macchine potenti oggi giorno vengono utilizzate come server di rete

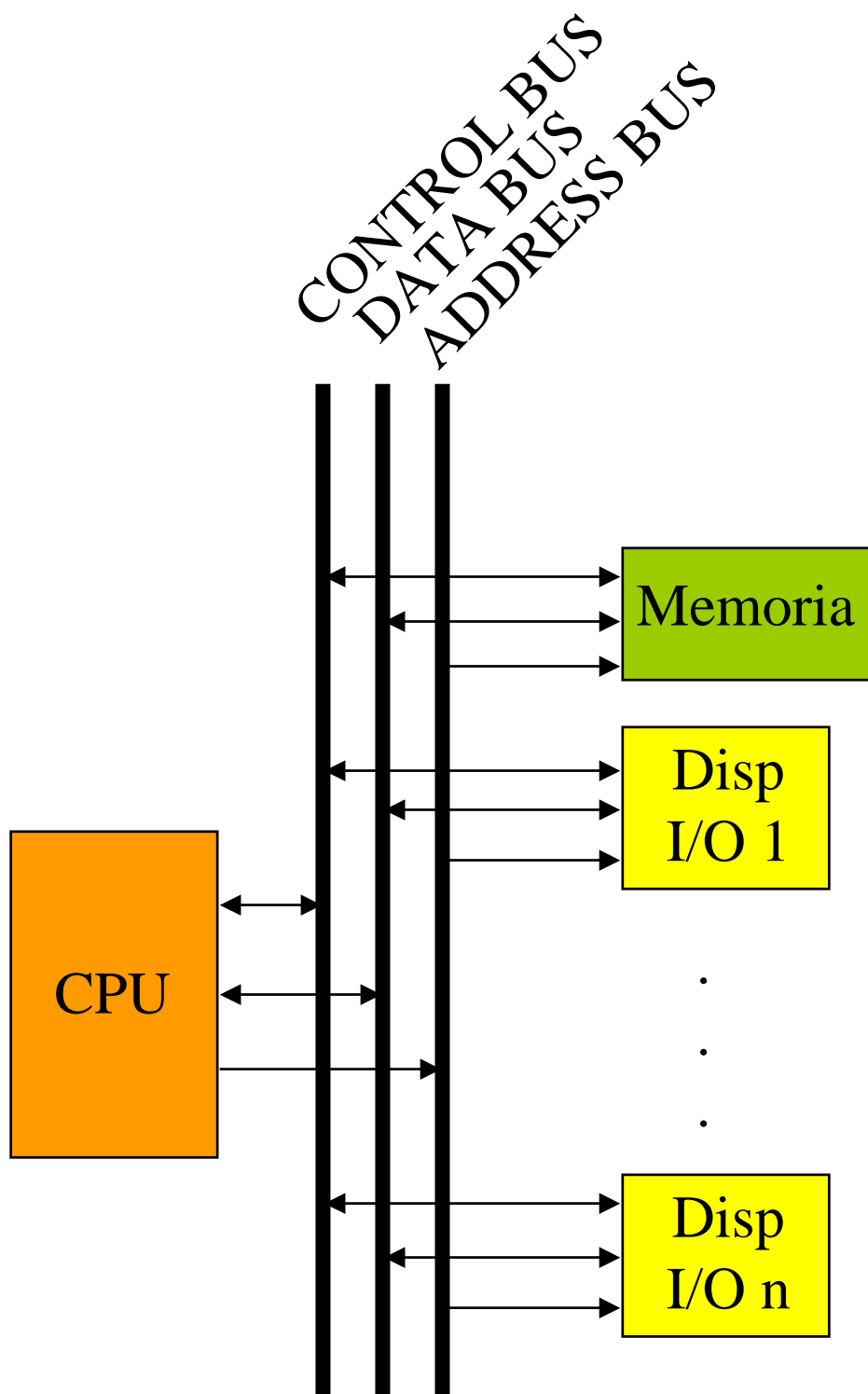
# Architettura di un sistema di elaborazione

---

- ◆ Il Personal Computer è la categoria di elaboratori più largamente diffusa e verrà trattata con maggior dettaglio nel seguito
- ◆ Non è possibile identificare un unico *schema fisico* (realizzativo) dell'architettura interna di un sistema di elaborazione, in quanto questa può variare notevolmente
- ◆ E' invece possibile identificare uno *schema logico* di massima che si adatta abbastanza bene per descrivere l'architettura della maggior parte dei SE attuali

# Architettura di un sistema di elaborazione

---

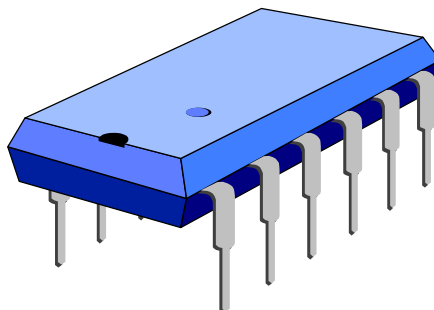




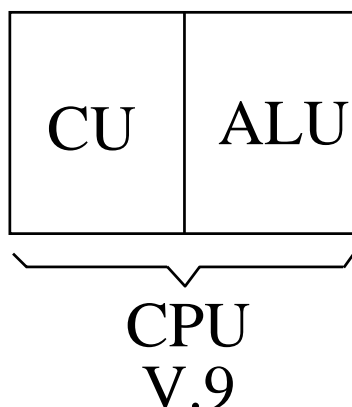
# Architettura interna di un sistema di elaborazione

---

- ◆ La **CPU** (*Central Processing Unit*) è il “cervello” del sistema; è la componente che è in grado di eseguire i programmi, fare i calcoli oltre a controllare le altre componenti del sistema



- ◆ La CPU può essere pensata come divisa logicamente in due parti
  - unità di controllo (CU)
  - unità aritmetico-logica (ALU)



# La CPU

---

- ◆ La CU ha il compito di coordinare l'attività interna della CPU stessa con quella di tutte le altre componenti del sistema
- ◆ La ALU ha il compito di eseguire i calcoli
- ◆ La CPU è ovviamente **molto** più complessa, però è importante sottolineare come concettualmente le attività che essa svolge sono due:
  - controllo delle attività del sistema
  - esecuzione delle operazioni

# La CPU

---

- ◆ La CPU mantiene al suo interno le informazioni da elaborare in appositi “contenitori” chiamati *registri*
- ◆ Il motivo per cui il processore mantiene, se possibile, le informazioni al suo interno è quello di aumentare la velocità di esecuzione: non deve andare a cercare i dati in memoria
- ◆ Alcuni registri possono avere un compito specifico mentre altri possono essere generici

# La CPU

---

- ◆ La CPU controlla tutto il sistema di elaborazione mandando e ricevendo segnali per mezzo del *control bus*:
  - ready, busy, write, read, mem/IO,...
- ◆ Un *bus* non è altro che una serie di fili (in realtà sono delle piste metalliche) ognuno dei quali può “trasportare” dei segnali elettrici digitali (assumono solo 2 valori di tensione → algebra di Boole!)

# La CPU

---

- ◆ La velocità di elaborazione dipende da più fattori, tra i quali:
  - l'architettura dell'elaboratore
    - » presenza di molti o pochi registri
    - » componenti più o meno veloci
    - » etc.
  - il programma eseguito
    - » alcune CPU sono particolarmente veloci ad elaborare valori interi
    - » etc.
  - la frequenza del segnale di clock
    - » il *clock* è il segnale di sincronizzazione di tutto il sistema, più è alta la frequenza più è veloce il sistema

# La CPU

---

- ◆ La frequenza  $f$  del clock si misura in MHz
- ◆ La CPU è in grado di eseguire un'istruzione (in media) ogni  $n$  colpi di clock
- ◆ Il numero di istruzioni eseguite per secondo sarà allora:  $f / n$
- ◆ Si misura in MIPS (Million Instructions Per Second)
- ◆ Esempio  
CPU con  $f = 100$  MHz e in media 1 istruzione ogni 2 colpi di clock:
  - allora  $100/2 = 50$  MIPS
- ◆ Misura poco affidabile

# La memoria

---

- ◆ I programmi, per essere eseguiti, devono essere presenti nel sistema di elaborazione, ovvero memorizzati in componenti dette *memorie*
- ◆ Le memorie sono dei contenitori di informazioni
- ◆ Poiché l'informazione elementare comprensibile da un calcolatore è il bit, le memorie contengono un certo numero di *celle* (variabile a seconda della dimensione della memoria) ognuna delle quali può contenere un bit

# La memoria

---

- ◆ L'unione di otto celle consecutive è detta *byte*
- ◆ Solitamente la *capacità di memorizzazione* (o *dimensione*) delle memorie è espressa in Mega byte; ad esempio: 16 Mbyte, 64 Mbyte, etc.

	1	0	1	0	0	0	1	0
BIT →	0	0	1	1	0	1	1	1
	0	0	0	0	1	1	1	0
	0	0	0	0	0	0	0	0
BYTE →	1	1	1	1	1	1	1	1
	0	0	0	0	1	1	1	1
	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1

MEMORIA



# La memoria

---

- ◆ Quando la CPU fa riferimento ad una cella si dice che la “indirizza”, ovvero la identifica con un “nome” numerico univoco tra tutte le celle
- ◆ Questo è il motivo per cui si parla di *indirizzi di memoria*
- ◆ In realtà la CPU, per motivi di efficienza, non accede mai ad una singola cella (bit) alla volta, ma a multipli di  $m$  byte: *locazione*
- ◆ Ad ogni indirizzo di memoria corrispondono  $m$  byte, ossia 1 locazione

# La memoria


---

- ◆ La CPU comunica alla memoria l'indirizzo della locazione che intende leggere o scrivere per mezzo dell'*address bus* (bus degli indirizzi)
- ◆ Avendo a disposizione  $n$  fili di indirizzo si possono avere  $2^n$  combinazioni  $\rightarrow 2^n$  indirizzi diversi  $\rightarrow 2^n \cdot m$  byte

# La memoria

---

- ◆ All'indirizzo più basso (cioè  $0_H$ ) farà riferimento la prima locazione di memoria, all'indirizzo  $1_H$  la locazione successiva e così via

Indirizzi		Locazioni							
	$FFFF_H$	1	1	1	1	1	1	1	1
	$FFFE_H$	1	1	1	1	1	1	1	0
	$FFFD_H$	1	1	1	1	1	1	0	1
	...								
	$0010_H$	0	0	0	0	0	0	1	0
	$0001_H$	0	0	0	0	0	0	0	1
	$0000_H$	0	0	0	0	0	0	0	0

# La memoria

---

- ◆ Il trasferimento dei dati dalla CPU alla memoria e viceversa avviene tramite il *data bus*
- ◆ La dimensione (il numero di fili) del data bus indica il *parallelismo* della memoria:
  - se il data bus ha 8 fili significa che si può scrivere o leggere dalla memoria un byte alla volta
  - se il data bus ha 16 bit possono essere effettuate operazioni di lettura e scrittura della memoria che coinvolgono 2 byte

# La memoria

---

- ◆ Il termine *word* indica il parallelismo del data bus della CPU
- ◆ E' possibile avere parallelismo della memoria a 16 bit (2 byte - spesso detta anch'essa *word*), 32 bit (4 byte - *double word* ) e a 64 bit (8 byte - *quad word*)

# La memoria

---

- ◆ Le memorie si dividono in due categorie principali:
  - RAM
  - ROM
- ◆ Le memorie **RAM** (*Random Access Memory*)
  - possono essere scritte e lette un numero illimitato di volte
  - sono dette *volatili* in quanto se il sistema viene spento il loro contenuto viene perso

# La memoria

---

## ◆ Le memorie ROM (*Read Only Memory*)

- scritte una volta sola dal costruttore
- i dati non sono persi togliendo l'alimentazione (non sono “*volatili*”)
- possono essere lette un numero illimitato di volte

Utilizzi:

- piccoli programmi di uso frequente
- istruzioni che il sistema di elaborazione deve eseguire ogni volta all'accensione

# La memoria

---

## Classificazione delle RAM

- **DRAM** (Dynamic RAM )

Sono le memorie più economiche e frequentemente usate. Sono memorie relativamente “lente” perché richiedono un continuo aggiornamento dei dati durante il quale la memoria non è utilizzabile

Quando si parla per un Personal Computer di 32 Mbyte di “RAM” si intende in realtà DRAM

- **SRAM** (Static RAM )

Più veloci (e costose) delle DRAM, non hanno bisogno del continuo aggiornamento dei dati



# La memoria

---

## Classificazione delle ROM

- **ROM** - Scritte dal costruttore
- **PROM** (Programmable ROM) - Possono essere scritte una volta sola con dispositivi speciali
- **EPROM** (Erasable PROM) - Possono essere riscritte solo alcune volte con dispositivi speciali (cancellazione a raggi UV)
- **EEPROM** (Electrically EPROM) - Come le precedenti, vengono cancellate con impulsi elettrici
- **Flash EPROM** - Riscrivibili più volte senza apparecchiature particolari; si possono scrivere solo a settori  $\Rightarrow$  non utilizzabili come DRAM, ma come piccole memorie di massa per dispositivi programmati (es. modem)

# La memoria

---

- ◆ Nei sistemi di elaborazione moderni, al fine di aumentare l'efficienza, le memorie sono organizzate in modo gerarchico
- ◆ L'obiettivo è quello di mettere diversi livelli di memoria. I livelli più vicini al processore sono più veloci di quelli più lontani
- ◆ Una maggior velocità da parte delle memorie implica anche un maggior costo, quindi, i livelli più vicini al processore hanno una capacità di memorizzazione inferiore rispetto a quelli lontani

# La memoria

---

- ◆ Tra la CPU e la memoria DRAM di sistema vengono “interposti” uno o due livelli di SRAM detta *memoria cache*
- ◆ La cache si classifica in:
  - *cache di I° livello*  
fisicamente situata nello stesso chip del processore (4-64 KByte)
  - *cache di II° livello*  
esterna al processore (256-2048 KByte) su un chip a parte

# La memoria

---

- ◆ Il principio che ha ispirato l'adozione delle cache è basato sulla *proprietà di località dei dati*: è molto probabile che la CPU faccia accesso a locazioni di memoria consecutive
- ◆ Al primo riferimento ad una locazione, non solo quella viene caricata nella cache, ma anche alcune locazioni ad essa consecutive
- ◆ Se la CPU chiede di accedere alle locazioni successive a quella appena richiesta, queste sono già pronte in cache

# La memoria

---

- ◆ Tanto più è alta la capacità di memorizzazione delle cache tanto maggiore è la probabilità che i riferimenti della CPU interessino dati contenuti in cache
- ◆ Quando la CPU richiede un dato non presente in cache si verifica un *cache miss*
- ◆ A seguito di un cache miss il dato richiesto (e quelli successivi) deve essere portato dalla memoria di sistema in cache (eventualmente eliminando dalla cache dei dati) prima di poter essere usato dalla CPU

# La memoria

---

- ◆ Una buona strategia di gestione della cache riduce la quantità di cache miss
- ◆ *Coerenza della cache:*  
quando la CPU modifica un dato presente in cache bisogna che la modifica avvenga anche nella memoria di sistema
- ◆ La strategia di gestione bilancia efficienza e sicurezza (in caso di mancanza improvvisa dell'alimentazione)

# Dispositivi di I/O

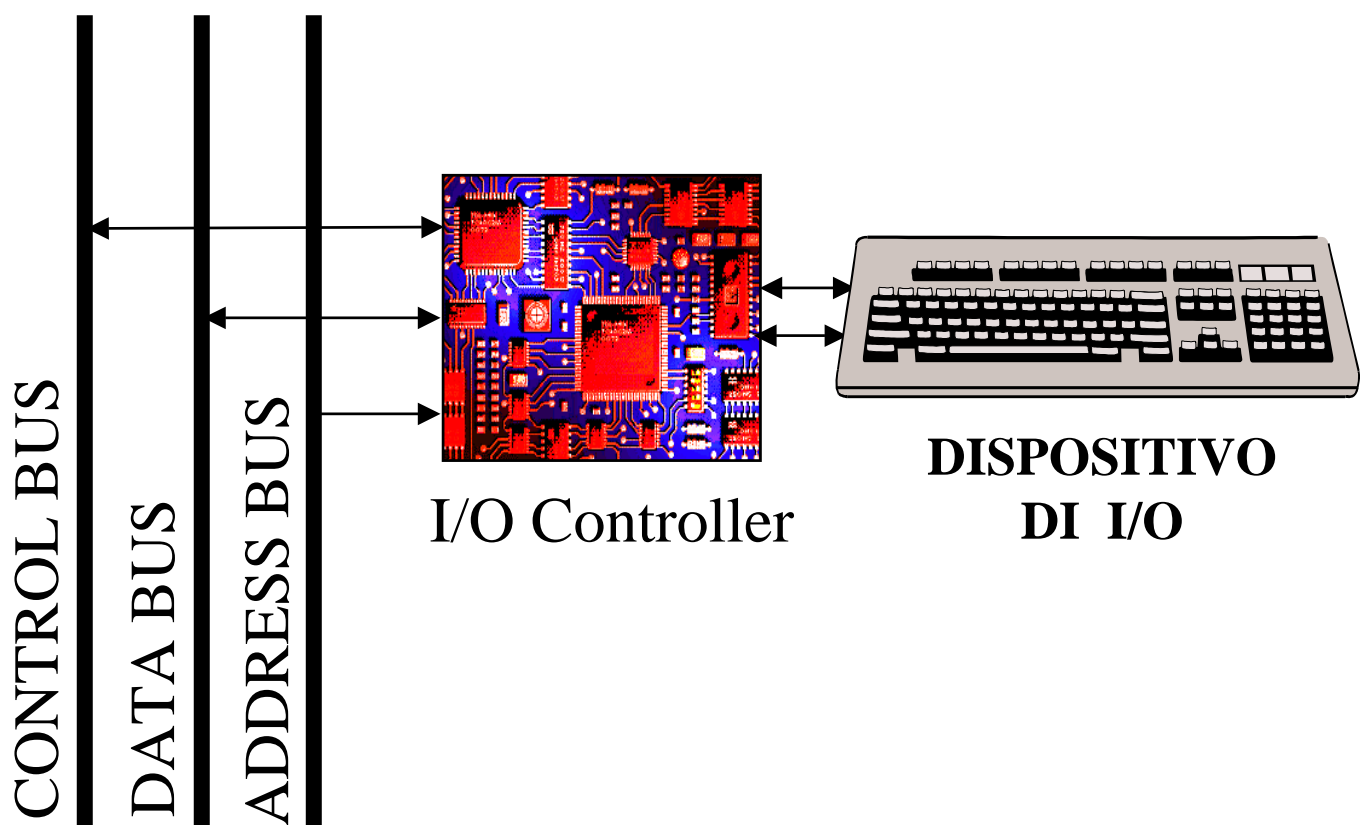
---

- ◆ Per interagire con il “mondo esterno” il sistema di elaborazione ha bisogno di opportuni dispositivi
- ◆ Tali componenti si dicono *dispositivi di Input-Output (I/O)*
- ◆ I dispositivi di **Input** servono per introdurre nel sistema di elaborazione, dal mondo esterno, dati o istruzioni
- ◆ I dispositivi di **Output** servono a trasferire verso l'esterno le informazioni elaborate dal calcolatore

# Dispositivi di I/O

---

- ◆ I dispositivi di I/O non sono gestiti direttamente dalla CPU, ma attraverso circuiti di controllo chiamati *I/O controller* (controllori di dispositivi di I/O)
- ◆ Lo schema tipico è il seguente:





# Dispositivi di I/O

---

- ◆ I *dispositivi di Input* sono generalmente composti da:
  - un’eventuale parte meccanica
  - una parte elettronica

- ◆ Esempio

La tastiera ha una parte “meccanica” costituita dai tasti. La pressione e il rilascio di un tasto fanno in modo che le componenti elettromeccaniche della tastiera trasmettano al sistema di elaborazione il codice ASCII del tasto appena premuto

# Dispositivi di I/O

---

- ◆ Un discorso analogo vale per i *dispositivi di Output* che sono costituiti, in generale, da una parte elettronica e da un'eventuale parte meccanica
- ◆ Esempio  
La stampante ha una componente elettronica che riceve dal sistema di elaborazione il codice ASCII del carattere da stampare e attiva la parte meccanica (es. la testina) affinché venga stampato il carattere corretto

# Dispositivi di I/O

## Gestione

---

- ◆ E' importante analizzare come i dispositivi periferici e la CPU interagiscono
- ◆ Quando la CPU vuole che un dispositivo periferico esegua una determinata operazione invia il comando corrispondente al *controller* del dispositivo
- ◆ Il controller interpreta il comando ricevuto e “pilota” il periferico (gli invia segnali elettrici di controllo) in modo che esegua il comando impartito dalla CPU

# Dispositivi di I/O

## Gestione

---

- ◆ La CPU può usare due tecniche differenti per accorgersi che il dispositivo periferico ha assolto al comando impartito:
  - gestione in *polling*
  - gestione in *interrupt*
- ◆ Con la *gestione in polling* (detta anche di attesa attiva) la CPU periodicamente va a controllare lo stato del dispositivo

# Dispositivi di I/O

## Gestione

---

- ◆ Nella gestione in polling può accadere che:
  - la CPU controlli per un certo numero di volte lo stato del dispositivo senza che questi abbia ancora terminato il suo compito: spreco di tempo di CPU che potrebbe essere utilizzato per svolgere altre attività
  - la CPU non si accorga immediatamente che il dispositivo ha terminato perché il controllo avviene periodicamente
- ◆ Inadatta per sistemi operativi multitasking dove la CPU deve essere sempre sfruttata al massimo
- ◆ Molto semplice da realizzare e non richiede hardware aggiuntivo

# Dispositivi di I/O

## Gestione

---

- ◆ Con la *gestione in interrupt* (interruzioni) la CPU assegna un compito ad un dispositivo e poi procede a svolgere altre attività
- ◆ Quando il dispositivo ha terminato il suo compito avverte la CPU mediante un segnale detto *interrupt*
- ◆ La CPU, a seguito di un interrupt, interrompe appena possibile la sua elaborazione attuale per “servire” il dispositivo periferico

# Dispositivi di I/O

## Gestione

---

- ◆ In realtà la CPU può decidere di interrompere le sue attività
  - in seguito a un qualunque interrupt
  - in seguito a interrupt selezionati
- ◆ Con la gestione mediante interrupt la CPU non spreca mai tempo per controllare inutilmente lo stato del dispositivo ed è immediatamente avvertita quando il periferico ha terminato il compito

# Dispositivi di I/O

## Gestione

---

- ◆ La gestione mediante interrupt è più complessa di quella in polling e richiede hardware aggiuntivo
- ◆ Le interruzioni provenienti dai dispositivi, in genere, non vanno direttamente alla CPU, ma ad un circuito chiamato *interrupt controller*
- ◆ La gestione mediante interrupt è in generale adatta a gestire *eventi asincroni* (ossia eventi “inattesi”)

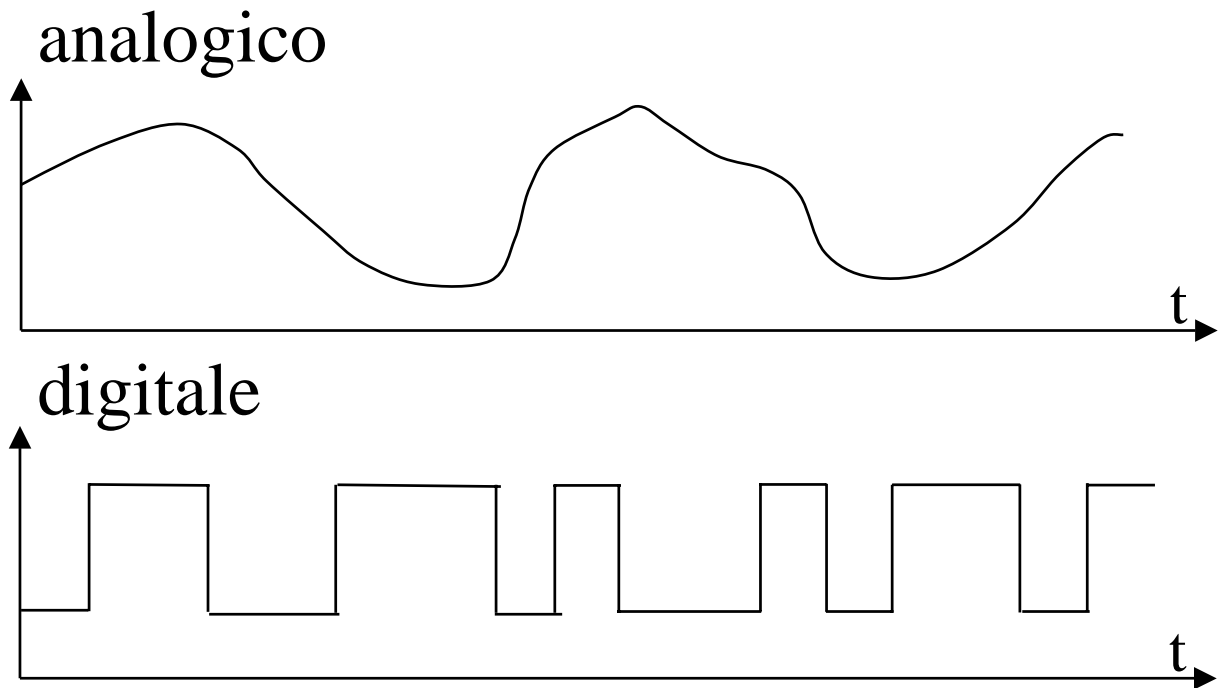


# Dispositivi di I/O

## Segnali

---

### ◆ Segnali elettrici:



- ◆ Un calcolatore è una macchina digitale e produce segnali digitali
- ◆ Per avere segnali analogici (e viceversa) si usano circuiti elettronici di conversione
- ◆ I circuiti di conversione si chiamano convertitori A/D e D/A

# Dispositivi di I/O

## Trasferimento dati

---

◆ Le **modalità di collegamento** di un periferico al calcolatore sono legate alle modalità di trasferimento dei dati:

- **Comunicazione seriale**

i bit vengono trasmessi uno per volta (ad esempio su un unico filo)

- **Comunicazione parallela**

più dati vengono trasmessi alla volta (ad esempio su più fili)

◆ **Esempi**

- La tastiera invia il codice ASCII del tasto premuto in modo seriale al controller

- La stampante riceve gli 8 bit del codice ASCII del carattere da stampare contemporaneamente su 8 fili diversi (uno per ciascun bit)

# Dispositivi di I/O

## Trasferimento dati

---

- ◆ Le modalità di comunicazione seriale sono regolamentate da un opportuno standard
- ◆ Le velocità tipiche delle comunicazioni seriali si misurano in bit trasmessi al secondo (bit/sec = baud in questo caso) e sono:

110 bit/sec	9600 bit/sec
150 bit/sec	14400 bit/sec
300 bit/sec	19200 bit/sec
600 bit/sec	28800 bit/sec
1200 bit/sec	57600 bit/sec
2400 bit/sec	115200 bit/sec
4800 bit/sec	

- ◆ I nuovi sistemi (USB) hanno velocità dell'ordine dei Mbit/sec

# Dispositivi di I/O

## Trasferimento dati

---

- ◆ Spesso, insieme ai dati veri e propri si trasmettono anche dei *bit di controllo*, così che il dispositivo in ricezione possa controllare la correttezza dei dati ricevuti
- ◆ Il metodo di controllo più semplice è il *bit di parità (parity bit)*
- ◆ Consiste nel trasmettere, dopo un byte di dati, un bit in più; il valore di questo bit è tale che:
  - nel caso di *parità dispari* il numero complessivo di 1 sia dispari
  - nel caso di *parità pari* il numero complessivo di 1 sia pari

# Dispositivi di I/O

## Trasferimento dati

---

### ◆ Esempio

Si vuole trasmettere il seguente byte con controllo di parità dispari:

01101100

- si trasmettono gli 8 bit di dato seguiti da un 1, in modo che in totale il numero di 1 sia dispari:

011011001

- ### ◆ Nel calcolare la velocità effettiva di comunicazione, bisogna considerare che il bit di parità non è un bit di dato e come tale “degrada” le prestazioni della comunicazione

# Dispositivi di I/O

## Trasferimento dati

---

### ◆ Esempio

Si devono trasmettere 1000 byte  
ad una velocità di 300 bit/sec  
adottando la tecnica del controllo  
di parità

- Ogni 8 bit ce n'è uno di parità,  
quindi si trasmettono 1000 blocchi  
di 9 bit:  $1000 \cdot 9 = 9000$  bit totali
- A 300 bit al secondo sono necessari  
30 secondi per la trasmissione

# Dispositivi di I/O

## Trasferimento dati

---

- ◆ Sincronizzazione di trasmettitore e ricevitore:
  - modalità *sincrona*: adottata per comunicazioni veloci, i bit vengono trasmessi senza interruzioni
  - modalità *asincrona*: adottata per comunicazioni lente ( $\leq 100$  Kbps), i dati vengono generalmente trasmessi a blocchi di 8 bit (*ottetti* coincidenti con i byte), ciascuno preceduto da un bit che indica che “sta per arrivare l’ottetto” (bit di START) e da 1, 1½ o 2 bit (bit di STOP, il numero è da specificare) che indicano che il byte è finito

# Dispositivi di I/O

## Trasferimento dati

---

- ◆ In una trasmissione asincrona è necessario considerare anche il tempo necessario per trasmettere i bit di START e di STOP
- ◆ Esempio Si devono trasmettere *in modo asincrono* (2 bit di stop) 1000 byte ad una velocità di 300 bps adottando la tecnica del controllo di parità. Tempo richiesto?
  - Ogni 8 bit se ne aggiungono:
    - » 1 di START
    - » 1 di parità
    - » 2 di STOP

Quindi si trasmettono 1000 blocchi di 12 bit:  
 $1000 \cdot 12 = 12000$  bit totali

A 300 bit al secondo sono necessari 40 secondi per la trasmissione



# Dispositivi di I/O

## Memorie di massa

---

- ◆ Come detto in precedenza le memorie RAM dei sistemi di elaborazione sono volatili e non consentono una memorizzazione dei dati anche dopo lo spegnimento del calcolatore
- ◆ Inoltre, la dimensione delle memorie RAM non sarebbe, in generale, tale da consentire di memorizzare tutti i dati e i programmi necessari ad un utente

# Dispositivi di I/O

## Memorie di massa

---

- ◆ I dispositivi che permettono di memorizzare grandi quantità di informazioni senza perdere il contenuto allo spegnimento vengono chiamati *memorie di massa*
- ◆ Per memorie di massa si intendono i dischi (hard disk, floppy disk, CD-ROM) e i nastri
- ◆ I dischi hanno la caratteristica di essere dei dispositivi sia leggibili che scrivibili (i CD-ROM sono solo leggibili)

# Dispositivi di I/O

## Memorie di massa

---

- ◆ I dischi consentono sia di introdurre delle informazioni nel sistema di elaborazione (si pensi al floppy) sia di poter ricevere delle informazioni dallo stesso
- ◆ Per questo motivo i dischi possono essere considerati dei periferici sia di Input che di Output
- ◆ Come per le memorie, la capacità di memorizzare informazioni di un disco è misurata in Mega o Giga byte. Valori tipici sono: 500 MB, 850 MB, 1 GB, 2 GB, 4 GB, 8 GB, etc.

# Principi di funzionamento dei sistemi di elaborazione

---

- ◆ Come detto in precedenza, la CPU è logicamente composta da due parti
  - unità di controllo
  - unità aritmetico-logica
- ◆ Il compito della CPU è quello di eseguire le “istruzioni”
- ◆ Le istruzioni sono le indicazioni delle operazioni che la CPU deve eseguire; queste possono essere di diversa natura:
  - aritmetiche
  - logiche
  - di I/O

# Principi di funzionamento dei sistemi di elaborazione

---

- ◆ In generale, un'istruzione specifica una serie di operazioni elementari da svolgere e quali dati devono essere considerati
- ◆ Un programma è una sequenza di istruzioni usata per risolvere un determinato problema
- ◆ Il programma indica quali sono le operazioni da svolgere e la CPU le esegue
- ◆ Nei primi calcolatori le istruzioni venivano inserite in modo meccanico, ad esempio, mediante schede perforate

# Principi di funzionamento dei sistemi di elaborazione

---

- ◆ Una prima evoluzione si ebbe quando si introdusse il concetto di programma immagazzinato nella memoria del calcolatore
- ◆ Istruzioni e dati sono quindi immagazzinati insieme in memoria, un'istruzione è codificata come sequenza di bit esattamente come lo è un dato
- ◆ La CPU sa quando il byte letto è un dato o un'istruzione

# Principi di funzionamento dei sistemi di elaborazione

---

- ◆ La parte della CPU che permette di “distinguere” tra dati e istruzioni si chiama **Program Counter (PC)**
- ◆ Il Program Counter è un registro che contiene l’indirizzo della locazione di memoria nella quale è memorizzata la prossima istruzione da eseguire
- ◆ Le locazioni che seguono possono contenere dati su cui operare, indirizzi di memoria contenenti dati oppure altre istruzioni

# Principi di funzionamento dei sistemi di elaborazione

---

- ◆ Il calcolatore, comunque, è a conoscenza della lunghezza dell'istruzione che sta eseguendo. Questo gli permette di aggiornare il Program Counter in modo che contenga sempre l'indirizzo di memoria della prossima istruzione da eseguire
- ◆ Il concepire i dati al pari delle istruzioni come sequenze di bit presenta il vantaggio di avere un'unica memoria (e non una per i dati e una per le istruzioni)



# Principi di funzionamento dei sistemi di elaborazione

---

- ◆ In generale, un'istruzione può essere composta da più parti (campi)
- ◆ La prima parte, sempre presente, è detta *codice operativo*
- ◆ Il codice operativo indica alla CPU quale istruzione deve essere eseguita
- ◆ Decodificando il codice operativo, la CPU conosce anche la lunghezza dell'istruzione e, quindi, può interpretare il significato delle parti restanti dell'istruzione medesima

# Principi di funzionamento dei sistemi di elaborazione

---

## ◆ Esempio

Se il codice operativo indica che deve essere eseguita una somma, le parti restanti dell'istruzione possono essere:

- i due operandi o gli indirizzi di memoria dove devono essere presi gli operandi
- l'indicazione di dove mettere il risultato

# Principi di funzionamento dei sistemi di elaborazione

---

- ◆ All'accensione del calcolatore, il Program Counter contiene sempre un indirizzo di memoria ROM prestabilito (solitamente la cella all'indirizzo 0)
- ◆ A partire da questa locazione di memoria vi è il programma che inizializza la macchina e i dispositivi e fa sì che il sistema operativo venga caricato in memoria dal disco
- ◆ La fase di caricamento di questo programma viene detta *bootstrap*

# Ciclo macchina

---

## Esecuzione di un'istruzione

- ◆ Questo meccanismo prende il nome di *ciclo macchina*
- ◆ La CPU conosce la posizione in memoria (l'indirizzo) della prossima istruzione da eseguire: è memorizzato nel Program Counter
- ◆ Fase di *fetch*
  - L'indirizzo di cui sopra viene messo dalla CPU sull'address bus
  - La memoria riceve l'indirizzo e mette sul data bus il contenuto della locazione indicata
  - La CPU legge il dato dal data bus e lo mette nel registro delle istruzioni

# Ciclo macchina

---

## ◆ Fase di *decode*

- La CPU esamina il contenuto del registro delle istruzioni (IR) e riconosce qual è l'istruzione che deve essere eseguita

## ◆ Fase di *execute*

- La CPU manda ai vari dispositivi (ALU inclusa) i comandi per eseguire l'istruzione indicata

# Ciclo macchina

---

- ◆ Le istruzioni possono essere composte da 1 o più parti
- ◆ La prima parte dell'istruzione è il *codice operativo* (sempre presente), questo permette alla CPU di:
  - capire quali sono le azioni richieste
  - aggiornare correttamente il PC affinché “punti” alla prossima istruzione da eseguire
- ◆ Se l'istruzione ha più di 1 parte, le successive sono i dati su cui operare

# Ciclo macchina

---

- ◆ I dati possono essere:
  - dei valori veri e propri
  - gli indirizzi di memoria dei valori
- ◆ In quest'ultimo caso la fase di fetch è più complessa in quanto la CPU dovrà:
  - mettere sull'address bus gli indirizzi degli operandi
  - leggere dal data bus gli operandi
- ◆ Quando la CPU dispone di tutti gli operandi può finalmente eseguire l'istruzione

# Sistemi operativi

---

- ◆ Il software può essere diviso in due classi:
  - i *programmi di sistema* che gestiscono le operazioni del sistema di elaborazione
  - i *programmi applicativi* che risolvono i problemi dei loro utilizzatori
- ◆ Il più importante dei programmi di sistema è il *sistema operativo* che controlla tutte le risorse del calcolatore e fornisce la base sulla quale possono essere sviluppate le applicazioni



# Sistemi operativi

---

- ◆ Un sistema di elaborazione moderno è costituito da numerosi dispositivi
- ◆ Incaricare il programmatore di gestire correttamente questi dispositivi risulterebbe difficile e inefficiente
- ◆ Il programmatore deve essere il più possibile svincolato dalla complessità dell'hardware
- ◆ Per ovviare al problema è necessario mettere uno strato di software tra l'hardware e l'utente

# Sistemi operativi

---

- ◆ Questo strato di software, chiamato sistema operativo, si presenta all'utente con una interfaccia o macchina virtuale più facile da:
  - capire
  - usare
  - programmare
- ◆ La situazione può essere rappresentata graficamente nel seguente modo:



# Sistemi operativi

---

- ◆ E' molto difficile descrivere esattamente cos'è un sistema operativo
- ◆ Il problema è dovuto al fatto che il sistema operativo svolge due funzioni tra loro scorrelate:
  - sistema operativo come *interfaccia uomo-macchina*
  - sistema operativo come gestore delle risorse
- ◆ La definizione come interfaccia tende ad evidenziare come il sistema operativo sia in grado di nascondere all'utente i dettagli dell'hardware

# Sistemi operativi

---

- ◆ La definizione come gestore delle risorse evidenzia la capacità del sistema operativo di gestire le risorse di sistema:
  - la CPU
  - la memoria
  - i dispositivi di I/O
  - etc.
- ◆ Prima di illustrare le tipologie dei sistemi operativi bisogna introdurre due concetti:
  - concetto di *utente*
  - concetto di *task*

# Sistemi operativi

---

## ◆ Concetto di *utente*

Nella terminologia dei sistemi operativi il termine utente indica una persona che utilizza il sistema di elaborazione

## ◆ Concetto di *task*

Quando un utente manda in esecuzione un programma chiede al sistema operativo di far svolgere all'hardware “un certo compito”. Questo compito viene detto task

# Sistemi operativi

---

- ◆ I sistemi operativi vengono generalmente classificati in:
  - mono-utente mono-tasking
  - mono-utente multi-tasking
  - multi-utente multi-tasking
- ◆ **I sistemi mono-utente mono-tasking** consentono ad un solo utente alla volta di utilizzare la macchina e l'utente non può mandare in esecuzione più task “simultaneamente”. Per poter eseguire uno nuovo task l'utente deve aspettare che il precedente sia terminato.
- ◆ Esempio: MS-DOS

# Sistemi operativi

---

- ◆ **Nei sistemi mono-utente multi-tasking** un solo utente può eseguire più task  
“contemporaneamente”
- ◆ Esempi: Windows 95 e NT
- ◆ **Nei sistemi multi-utente multi-tasking** più persone contemporaneamente possono utilizzare la macchina e ognuno può mandare in esecuzione più task
- ◆ Esempi: Unix, VMS, ...

# Sistemi operativi

---

- ◆ Non è possibile individuare un unico schema realizzativo per tutte le tipologie di sistemi operativi, però si possono identificare cinque componenti funzionali comuni a tutti i sistemi:
  - gestore della CPU
  - gestore della memoria
  - gestore dei dispositivi di I/O
  - gestore del file system
  - interprete dei comandi



# Sistemi operativi

## Gestore della CPU

---

- ◆ Il **gestore della CPU** è quel modulo del sistema operativo che ha il compito di decidere a quale *task* (non a quale *utente*) spetta l'assegnazione della CPU
- ◆ Tale componente prende spesso il nome di *scheduler* e gli algoritmi di scheduling sono le strategie adottate per assegnare la CPU ai task

# Sistemi operativi

## Gestore della memoria

---

- ◆ Il **gestore della memoria** è quel modulo del sistema operativo incaricato di assegnare la memoria ai vari task (per eseguire un task è necessario che il suo codice sia caricato in memoria)
- ◆ La complessità del gestore della memoria dipende dal tipo di sistema operativo
- ◆ Nei sistemi multi-tasking più programmi contemporaneamente possono essere caricati in memoria

# Sistemi operativi

## Gestore della memoria

---

- ◆ Spesso la memoria non è sufficiente per contenere completamente tutto il codice dei vari task
- ◆ Il disco però è molto capiente, si può *simulare* una memoria più grande tenendo nella memoria di sistema (RAM) solo le parti di codice e dei dati che servono in quel momento, lasciando sul disco tutto il resto: è il concetto di *memoria virtuale*
- ◆ Lo scambio di dati tra memoria fisica (RAM) e il disco è chiamato *swap* (scambio)

# Sistemi operativi

## Gestore dei dispositivi di I/O

---

- ◆ Il **gestore dei dispositivi di I/O** è quel modulo del sistema operativo incaricato di assegnare i dispositivi ai task che ne fanno richiesta
- ◆ In particolare deve gestire i *conflitti*, ovvero le situazioni in cui due o più task vogliono accedere contemporaneamente allo stesso dispositivo

# Sistemi operativi

## Gestore del file system

---

- ◆ Il **gestore del file system** è quel modulo del sistema operativo incaricato di gestire le informazioni memorizzate sui dispositivi di memoria di massa
- ◆ Il gestore del file system deve garantire la correttezza e la coerenza delle informazioni
- ◆ Inoltre, nei sistemi multi-utente, deve mettere a disposizione dei meccanismi di protezione in modo tale da consentire agli utenti di proteggere i propri dati dall'accesso da parte di altri utenti non autorizzati

# Sistemi operativi

## Interprete dei comandi

---

- ◆ **L'interprete dei comandi** è la vera interfaccia del sistema operativo verso l'utente
- ◆ Riceve i comandi dall'utente (generalmente da tastiera) e li esegue
- ◆ Nei sistemi multi-tasking l'utente può mandare in esecuzione un comando senza aspettare che il precedente sia terminato

# Istruzioni e linguaggi

---

- ◆ Come detto, le istruzioni al pari dei dati sono sequenze di bit
- ◆ Un programmatore può quindi scrivere i suoi programmi direttamente come sequenze binarie (*codice macchina*)
- ◆ Poiché le istruzioni che la CPU mette a disposizione sono molteplici, risulta molto difficile scrivere un programma in codice macchina

# Istruzioni e linguaggi

---

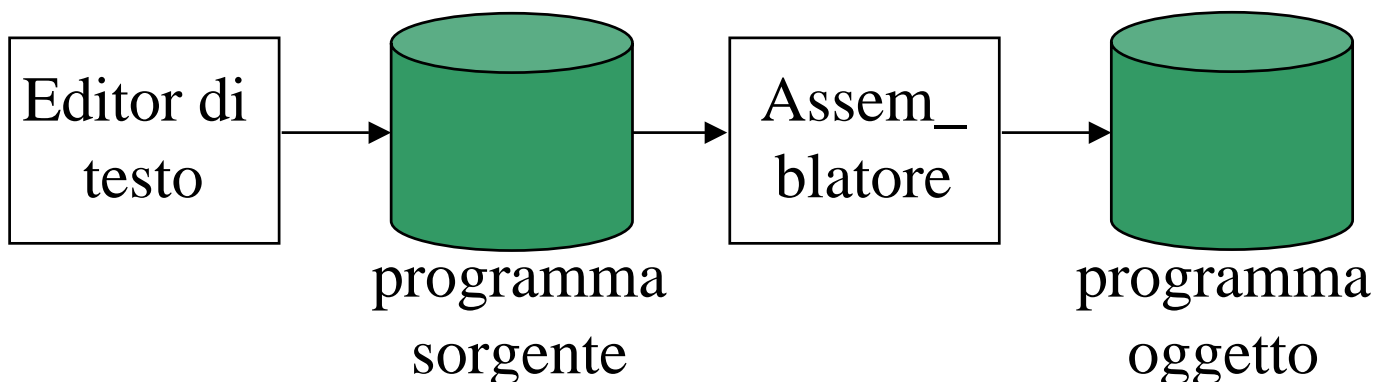
- ◆ Al fine di rendere più agevole la stesura dei programmi si è deciso di assegnare a istruzioni, dati e indirizzi dei nomi simbolici
- ◆ Ad esempio l'istruzione di somma potrebbe avere il nome simbolico ADD invece di 01001010
- ◆ Questo simbolismo è chiamato *linguaggio ASSEMBLY* (o *ASSEMBLER*)
- ◆ Per tradurre un programma scritto in assembly in codice macchina (che è l'unico linguaggio compreso dal calcolatore) viene utilizzato un programma chiamato *assemblatore* (*assembler*)



# Istruzioni e linguaggi

---

- ◆ L'assemblatore traduce i nomi simbolici delle istruzioni, dei dati e degli indirizzi assembly nei corrispondenti valori numerici
- ◆ L'operazione di stesura di un programma diventa quindi organizzata nelle seguenti fasi:
  - scrittura del programma assembly (*codice sorgente*) con un editor
  - traduzione del codice sorgente con l'assemblatore in codice macchina (*programma oggetto*)
  - esecuzione del programma oggetto



# Istruzioni e linguaggi

---

- ◆ Importante: il linguaggio assembly (e il linguaggio macchina) è diverso per ogni tipo di CPU
- ◆ Ad esempio, il linguaggio assembly dei processori Intel 80x86 (Personal Computer) non è compatibile con quello dei processori Motorola (Macintosh) o Digital
- ◆ Quindi i programmi scritti in assembly per una determinata macchina non possono essere trasportati su calcolatori di tipo diverso

# Istruzioni e linguaggi

---

- ◆ Al fine di ovviare a questo inconveniente è preferibile utilizzare *linguaggi ad alto livello*
- ◆ Nei linguaggi ad alto livello si hanno istruzioni con un più alto potere espressivo e, soprattutto, indipendenti dal tipo di CPU sulle quali devono essere eseguite

Esempio (stampa del numero 12):

- alto livello:    `PRINT 12`
- assembly:     `MOV AX,12`  
                  `INT 1F`

# Istruzioni e linguaggi

---

- ◆ Un programma scritto in un *linguaggio ad alto livello* deve essere convertito in *linguaggio macchina* per essere eseguito
- ◆ La traduzione può avvenire:
  - *riga per riga*: ogni riga di codice sorgente viene letta, convertita in linguaggio macchina e poi eseguita da un programma detto *interprete*
  - *tutto insieme*: tutto il codice viene letto e convertito in linguaggio macchina da un programma detto *compilatore*, l'esecuzione avviene in un momento successivo in seguito ad una richiesta esplicita dell'utente

# Istruzioni e linguaggi

---

◆ I più noti linguaggi ad alto livello sono i seguenti:

- C                      C++
- Fortran              Pascal
- BASIC                ADA
- LISP                  PROLOG
- COBOL                Java
- Perl                  APL
- PL/1                  RPG
- Smalltalk            Simula
- *e tanti tanti* altri

◆ Linguaggi interpretati:

- maggior controllo nella fase di sviluppo del programma
- minor velocità di esecuzione

# Istruzioni e linguaggi

---

- ◆ Più in dettaglio, un linguaggio compilato attraversa 2 fasi:
  - la fase di compilazione vera e propria
  - la fase di link
- ◆ Il compilatore genera un codice detto *rilocabile*, questo è costituito dalla traduzione delle istruzioni in linguaggio macchina, ma l'indicazione degli indirizzi di memoria è ancora lasciata in sospeso

*Perché?*

# Istruzioni e linguaggi

---

- ◆ Molto spesso programmi molto lunghi vengono suddivisi in moduli più corti, più facili da sviluppare e controllare, ma poi questi devono essere accorpati per costituire un unico programma eseguibile
- ◆ Lo strumento che mette insieme i vari moduli di un programma si chiama *linker*

# Istruzioni e linguaggi

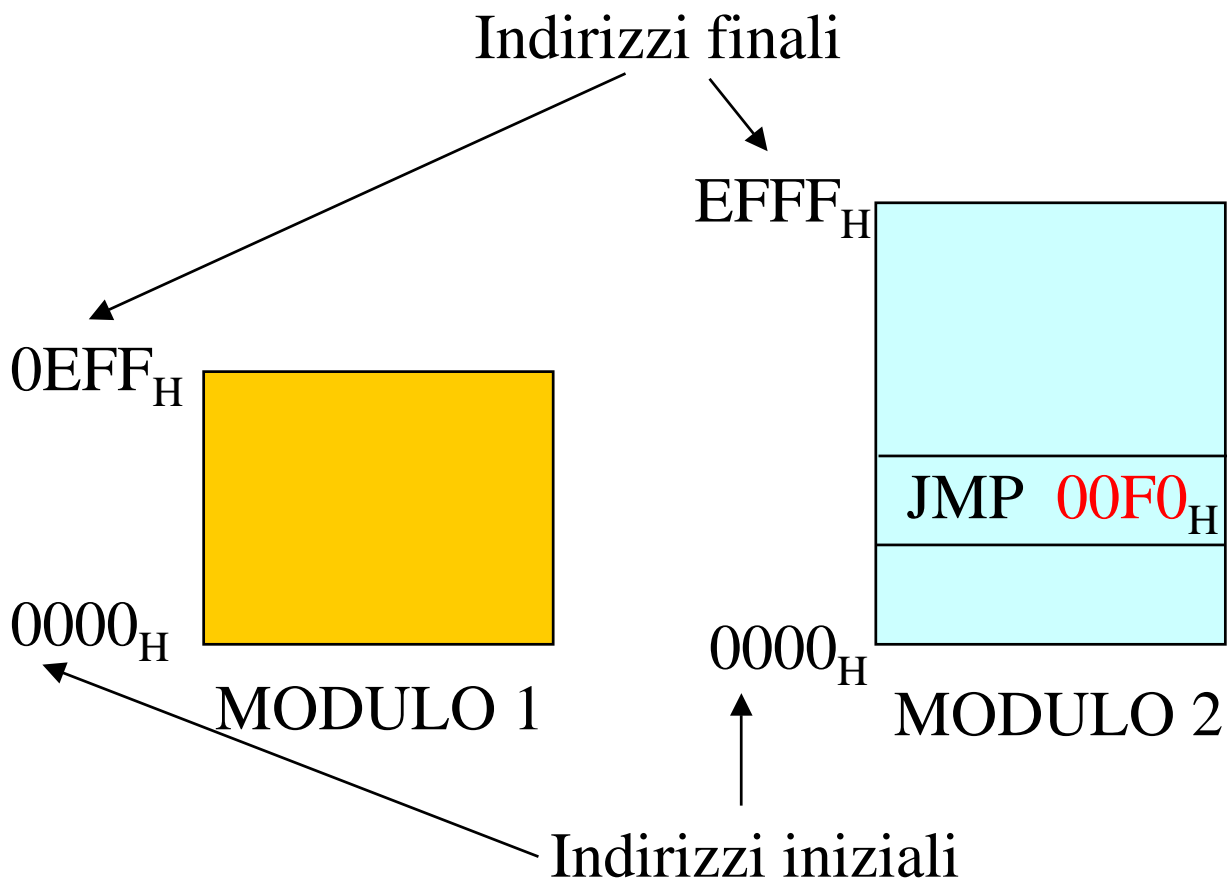
---

- ◆ Il linker risolve la rilocazione degli indirizzi. Infatti, se all'interno di un modulo c'è un riferimento ad un indirizzo, questo riferimento deve essere modificato in funzione della posizione del modulo dopo la sua unione con gli altri moduli



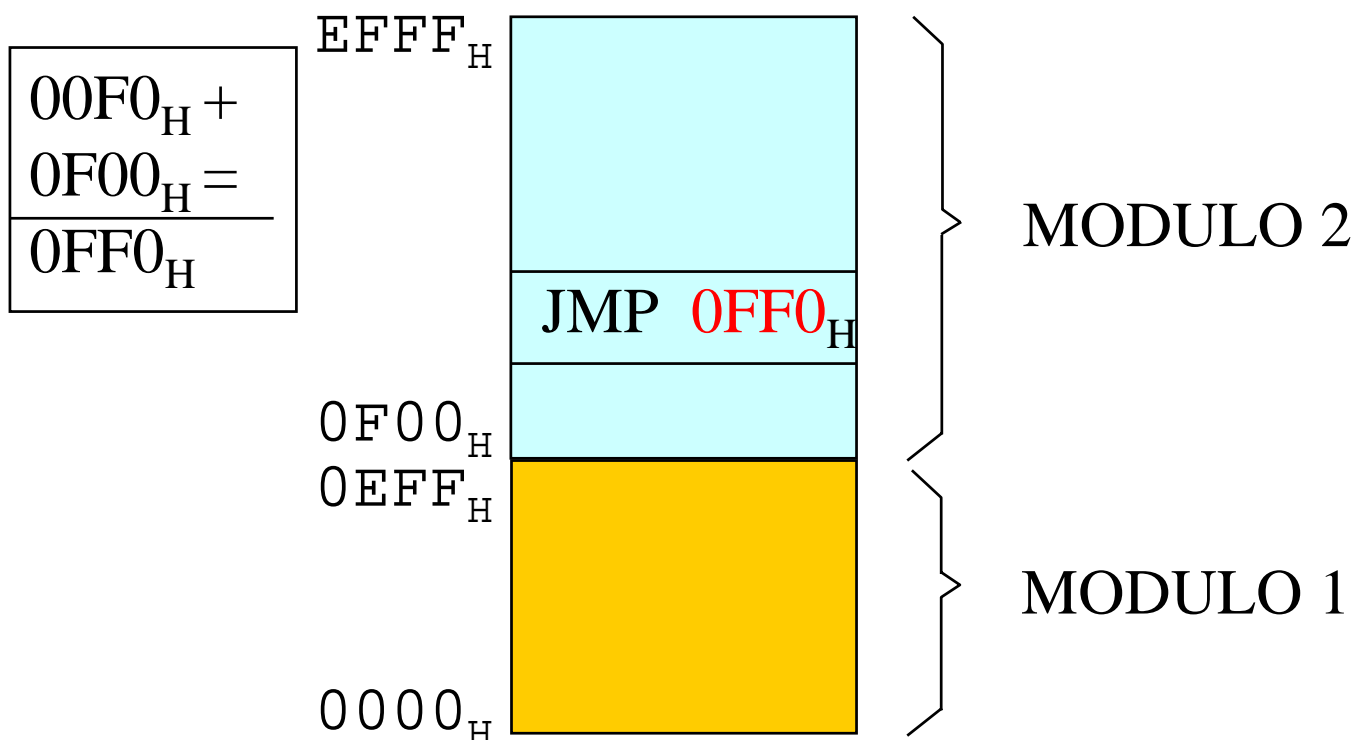
# Istruzioni e linguaggi

- ◆ Esempio Si supponga che il linker debba unire i due moduli di figura e nel secondo modulo c'è una istruzione di salto all'interno del modulo stesso



# Istruzioni e linguaggi

- ◆ Il linker deve unire i due moduli e rilocare gli indirizzi
- ◆ Il codice prodotto dal compilatore non è un semplice codice macchina, ma è qualcosa in più: contiene le informazioni per il linker



$00F0_H$  è rilocato più su di  $0F00_H$  locazioni

# Esercizi

---

- ◆ Esercizio 1 Quanti byte di memoria si possono indirizzare avendo a disposizione un data bus a 16 bit e un address bus a 10?
  - Con  $n$  fili di address bus si possono indirizzare  $2^n$  locazioni. Con 10 fili è possibile indirizzare  $2^{10} = 1024$  locazioni, ogni locazione contiene 2 byte, quindi:  $1024 \cdot 2 = 2048$  byte (2KB)
- ◆ Esercizio 2 Sia data una memoria di 80 KByte e di parallelismo 1 byte, quanti fili di indirizzamento sono necessari per indirizzare la memoria?

$$2^n \geq 80 \cdot 1024 \Rightarrow$$

$$n = \lceil \log_2(80 \cdot 1024) \rceil = \lceil 16,32 \rceil = 17$$

# Esercizi

---

◆ Esercizio 3 Supponendo di avere una memoria di 4 KB divisa in 4 banchi di 1 KB ciascuno, in quali parti della memoria risiedono i seguenti indirizzi?

a)  $001_H$

b)  $FFF_H$

c)  $900_H$

– Per indirizzare una memoria di 4 KB sono necessari 12 fili di indirizzo, infatti:

$$2^n \geq 4 \cdot 1024 \Rightarrow$$

$$n = \lceil \log_2(4 \cdot 1024) \rceil = \lceil 12 \rceil = 12$$

# Esercizi

---

Si può quindi supporre di dividere la memoria in 4 parti i cui limiti sono:

Banco I	$000_H$	← Primo indirizzo banco I
	$3FF_H$	← Ultimo indirizzo banco I
Banco II	$400_H$	← Primo indirizzo banco II
	$7FF_H$	← Ultimo indirizzo banco II
Banco III	$800_H$	← Primo indirizzo banco III
	$BFF_H$	← Ultimo indirizzo banco III
Banco IV	$C00_H$	← Primo indirizzo banco IV
	$FFF_H$	← Ultimo indirizzo banco IV

L'indirizzo  $001_H$  si riferisce ad una locazione nel primo banco,  
l'indirizzo  $FFF_H$  ad una locazione nell'ultimo banco e l'indirizzo  $900_H$  ad una locazione nel terzo banco

# Esercizi

---

◆ Esercizio 4 Si supponga di dover trasferire 16 KB su una linea seriale in meno di 8 secondi (senza tenere conto del bit di parità):

- a) quale deve essere la minima velocità di trasmissione della linea?
- b) quale è la velocità di trasmissione standard che meglio si adatta alle richieste?

a) Bisogna trasferire  $16 \cdot 1024 \cdot 8$  bit in 8 secondi quindi la velocità della linea deve essere di almeno 16 Kbps

b) La velocità di trasmissione che meglio si adatta è di 19200 Kbps