



## Tema 1

# Un nuovo strumento: l'Intelligenza Artificiale in Internet

## Che cos'è l'Intelligenza Artificiale?

L'Intelligenza Artificiale (IA) o Artificial Intelligence (AI) è una tecnologia informatica che consente alle macchine di imparare dall'esperienza, riproducendo i processi di apprendimento del nostro cervello.

Il nostro cervello riconosce per esempio un pallone perché si basa sulle conoscenze accumulate giocando con una palla, fin dall'infanzia. Le informazioni che ha appreso nel tempo gli permettono di individuare una struttura di base e riconoscere che un pallone giallo e uno bianco sono entrambi palloni perché sa distinguere le caratteristiche di primaria importanza da elementi variabili che non sono determinanti. Il linguaggio utilizzato nell'AI è lo stesso di altri programmi, ma gli algoritmi sono strutturati in modo diverso secondo un processo di addestramento («training») che consiste nell'insegnare alla macchina a interpretare i dati in arrivo in ingresso, estrarre le caratteristiche, confrontarle con i modelli presenti nella sua memoria, comprendere di cosa si tratta. Ogni volta che il processo di training è superato la macchina rafforza e aggiorna le

proprie conoscenze ed eviterà di ripetere eventuali errori commessi in precedenza.

L'AI è una tecnologia che nasce negli anni Cinquanta del secolo scorso, con il Test di Turing, ed è ancora in evoluzione.

Oggi i sistemi di AI si alimentano di una enorme quantità di dati (si parla appunto di «big data») che, per la maggior parte, siamo noi stessi a fornire attraverso le nostre abitudini e scelte nella rete. Ad esempio, ogni volta che accettiamo «tutti i cookies» di una pagina Web o facciamo una traduzione online, contribuiamo ad alimentare l'immenso serbatoio dei big data.

Questo ci fa anche riflettere sul fatto che in molti casi i servizi che la rete offre non sono affatto gratuiti. Per utilizzarli, nella maggior parte dei casi dobbiamo fornire il consenso all'utilizzo di alcuni nostri dati, oppure al tracciamento del nostro traffico in Internet. Questi dati sono usati ad esempio per personalizzare la pubblicità che vediamo. Perciò i servizi apparentemente gratuiti vengono in realtà pagati concedendo l'utilizzo dei nostri dati.

## Elementi chiave dell'AI

**Big data:** l'AI si alimenta di tantissimi dati (da qui la parola big) che, per essere informativi, devono essere eterogenei, affidabili ed elaborati velocemente.

**Machine Learning:** l'AI si basa su sistemi che apprendono o migliorano in base ai dati che utilizzano. Vendono usati due tipi di algoritmi: machine learning supervisionato e non supervisionato, in base al modo in cui l'algoritmo apprende i dati per fare previsioni. Il ML è utilizzato ovunque: acquisiti on line, social media, home banking, classificazione delle immagini, motori di raccomandazione...

**Reti neurali:** ricalcano il modo in cui i neuroni si scambiano i segnali e costituiscono l'elemento centrale degli algoritmi di **deep learning**, i quali utilizzano più livelli di elaborazione per garantire maggiore precisione ed efficacia.

**Data mining:** è una tecnica di analisi per reperire informazioni da grandi quantità di dati.

## Attività per lo studente

### Obiettivi

Ti proponiamo alcune attività che ti consentiranno di

- comprendere l'evoluzione che ha portato agli attuali sistemi di AI;
- riflettere sull'impatto dell'AI sul nostro modo di studiare e imparare.

**1.** Approfondisci i seguenti argomenti, anche lavorando in collaborazione con le tue compagne e i tuoi compagni di classe.

- Storia dell'AI
- Vantaggi e rischi dell'uso dei sistemi di AI
- Importanza della scelta dei dati per alimentare i sistemi di AI
- Fonti dei dati utilizzati per alimentare i sistemi di AI
- Ambiti di utilizzo dell'AI
- Modelli predittivo, prescrittivo, descrittivo

**In base alle tue/vostre ricerche, rispondi ora alle seguenti domande, utili per imparare a usare i sistemi di AI in modo consapevole.**

2. Con quale criterio e da chi sono selezionati i dati che alimentano i sistemi di AI?
3. In quali situazioni i sistemi di AI ti hanno realmente aiutato? In quali situazioni invece non si sono rivelati utili?
4. In quali modi l'utilizzo dei sistemi di AI influisce sulla nostra libertà, anche di sbagliare?
5. Fino a che punto il modo di apprendere di una macchina è paragonabile al tuo e, in genere, a quello delle persone?



## Tema 1

# Interroghiamo l'Intelligenza Artificiale

**Tu e un tuo compagno o una tua compagna scegliete 5 domande dall'elenco.**

**Successivamente ogni elemento della coppia, lavorando individualmente, interroga l'AI. I due membri della coppia utilizzano due chatbot diversi. Ciascuno conserva le schermate con le risposte e riassume le risposte ottenute.**

**Infine, confrontate le risposte. I chatbot hanno dato risposte analoghe? Oppure ci sono state differenze? Quando tu e il tuo compagno o la tua compagna avete riassunto le risposte del chatbot, avete considerato come importanti gli stessi concetti?**

1. Quali sono i componenti fondamentali di un sistema di elaborazione dati?
2. Che cos'è l'architettura di Von Neumann e come si applica ai computer moderni?
3. Qual è il ruolo della CPU in un sistema di elaborazione?
4. Descrivi la differenza tra memoria RAM e memoria ROM.
5. Elenca alcune delle principali periferiche di input e output presenti in un computer.
6. Che cosa si intende per «software» in un contesto di sistemi informatici?
7. Come è evoluta la tecnologia dei microprocessori nel corso degli anni?
8. Perché è importante comprendere i sistemi di numerazione binario ed esadecimale?
9. Quali sono gli elementi base della programmazione?
10. Che cosa sono le strutture di controllo in un linguaggio di programmazione ad alto livello?
11. Come definiresti il concetto di «architettura di un generico microprocessore»?
12. Qual è la differenza tra RAM e ROM, e come esse vengono utilizzate in un computer?
13. Che cos'è la gerarchia di memoria e perché è importante nei sistemi informatici?
14. Spiega il concetto di «stack» in informatica e come viene utilizzato.
15. Che cosa rappresenta il formato delle istruzioni nel linguaggio assembly?
16. Qual è il ruolo di un set di istruzioni in un processore?
17. Che cosa significa «programmare in linguaggio assembly» e quali sono i vantaggi?
18. Descrivi l'importanza delle reti e degli strumenti informatici nelle attività di studio e ricerca.
19. Come influisce la capacità della memoria su un sistema di elaborazione?
20. Che cosa significa «classificazione delle memorie» e quali sono le categorie principali?
21. Qual è l'importanza di una gerarchia di memoria ben strutturata in un sistema informatico?
22. In che modo le istruzioni nel linguaggio assembly differiscono da quelle in un linguaggio ad alto livello?

23. Come viene scelta l'architettura del set di istruzioni in un microprocessore?
24. Come puoi valutare le prestazioni di un sistema a microprocessore?
25. Qual è il ruolo dell'accessibilità nei sistemi informatici e nell'Educazione civica?
26. In quale modo i sistemi informatici possono contribuire alla tutela della privacy degli individui?
27. Qual è l'importanza dell'alfabetizzazione digitale nell'Educazione civica?
28. In quale modo i sistemi informatici possono essere utilizzati per promuovere la partecipazione civica e la democrazia?
29. Che cosa sono i rischi legati alla sicurezza informatica e come possono essere affrontati?
30. Quali sono le principali leggi e i principali regolamenti che governano l'uso dei sistemi informatici?
31. In che modo i sistemi informatici possono influire sulle questioni di diritti umani?
32. Qual è il ruolo della tecnologia nell'istruzione e nell'accesso all'istruzione?
33. In quale modo l'Educazione civica può essere promossa attraverso l'uso di tecnologie digitali?
34. Quali sono le implicazioni etiche e sociali dell'Intelligenza Artificiale e dell'automazione?
35. In quale modo i sistemi informatici possono contribuire a combattere la disinformazione e le fake news?
36. In quale modo i sistemi informatici possono migliorare la partecipazione politica e la responsabilità civica?

# Tema 2

# Utilizzo dei sistemi di elaborazione

## Unità

- 6 Ambienti di elaborazione**
- 7 L'architettura del microprocessore 8086**
- 8 L'architettura ARM**
- 9 Il Physical Computing**

### Prerequisiti

- Sistema di numerazione binario e sistema di numerazione esadecimale
- Conoscenza del significato di sistema operativo
- Conoscenza dell'architettura di un personal computer
- Elementi base di programmazione
- Elementi base di progettazione di circuiti elettronici

### Conoscenze

- Il physical computing
- La scheda madre di un PC
- I System On a Chip
- I Single Board Computer
- I microcontrollori
- Il processore 8086: architettura e istruzioni
- Il processore ARM: architettura e istruzioni
- IoT e mondo interconnesso
- Sensori e attuatori
- Arduino
- Raspberry

### Abilità

- **Individuare** le componenti di una scheda madre
- **Assemblare** un personal computer
- **Individuare** la corretta configurazione di un sistema
- **Identificare** i principali dispositivi periferici
- **Riconoscere** la struttura di un'istruzione in linguaggio di basso livello
- **Programmare** in assembly 8086 e ARM
- **Progettare** semplici sistemi del Physical Computing

### Competenze

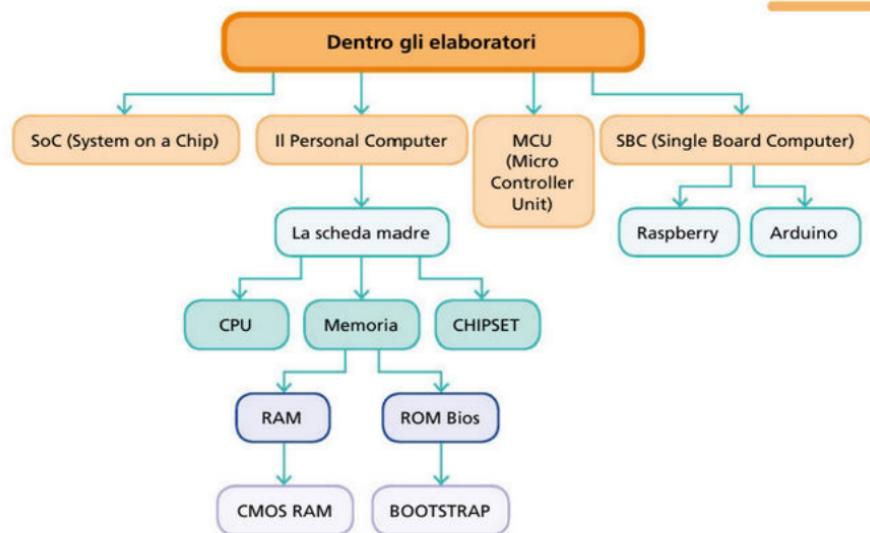
- **Configurare, installare e gestire** sistemi di elaborazione dati
- **Scegliere** dispositivi e strumenti in base alle loro caratteristiche funzionali
- **Utilizzare** Arduino e Raspberry
- **Utilizzare** software per la simulazione di ambienti reali
- **Descrivere e comparare** il funzionamento di dispositivi e strumenti elettronici e di telecomunicazione
- **Utilizzare** le reti e gli strumenti informatici nelle attività di studio, ricerca e approfondimento disciplinare





# Unità 6

## Ambienti di elaborazione



### Visione d'insieme

- Modelli di sistemi di elaborazione: dal Personal Computer (PC) al Single Board Computer (SBC).
- Architettura di un Personal Computer.
- Due esempi di SBC: Raspberry Pi e Arduino.

## 1 Introduzione

Nel corso degli anni la dimensione e la forma dei computer sono cambiati grazie all'evoluzione della tecnologia, con la conseguente riduzione degli spazi e dei consumi.

Per garantire la compatibilità delle diverse famiglie di elaboratori, per i componenti dei computer (schede, banchi di memoria, alimentatori, connettori ecc.) i produttori seguono regole di standardizzazione che stabiliscono i **fattori di forma** che determinano il design, le dimensioni e l'aspetto fisico di un hardware. L'esempio più evidente è dato dalla differenza tra un computer desktop e un portatile: entrambi sono costituiti dalle stesse parti realizzate in modo diverso. I fattori di forma di una scheda madre di un **PC desktop** impongono precise caratteristiche anche ai banchi di memoria e alle schede aggiuntive che possono essere inserite.

Considerazioni simili valgono anche per i chip che contengono l'intero sistema (**SoC - System-on-a-Chip**) e i *computer a scheda singola* (**SBC - Single Board Computer**), di cui parleremo alla fine dell'unità.

In questa Unità troveremo moltissimi acronimi, e ogni volta ne spiegheremo il significato.

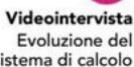
## 2 Il Personal Computer

Un **Personal Computer (PC)** è pensato per essere utilizzato da un singolo utente per eseguire applicazioni software.

Esso può assumere la configurazione di:

- **desktop**, se è utilizzato in una postazione fissa;
- **notebook** (o **laptop**), se è utilizzato in mobilità. In questo caso il sistema è integrato in un unico contenitore di dimensioni e peso ridotti per essere facilmente trasportato. Un notebook integra monitor, tastiera e batteria per l'alimentazione, ma la logica di funzionamento è la stessa di un desktop.

Di seguito presentiamo le componenti fondamentali di un desktop.



**Videointervista**  
Evoluzione del  
sistema di calcolo

### Pit Stop

- 1 Perché i fattori di forma di un hardware devono seguire delle regole di standardizzazione?
- 2 Qual è la differenza fondamentale fra un PC desktop e un notebook?

## 3 Il case

L'hardware di un desktop è inserito in un contenitore in metallo e plastica, il **case** (Figura 1), in cui sono posizionati i principali componenti:

- l'**alimentatore**, che fornisce l'energia elettrica necessaria per il funzionamento di tutte le parti del sistema;



↑ Fig. 1 Il case (modello *Mid Tower*) di un PC con scheda madre ATX.



↑ Fig. 2 Installazione di un *blade server*, cioè di un server a scheda singola.

- la **scheda madre**, con la **CPU**, le **memorie** e le **schede aggiuntive**.

Il case è progettato per favorire la dispersione del calore prodotto dal funzionamento della CPU e degli altri componenti, per evitarne il surriscaldamento. Il case può avere dimensioni diverse: dai 70 cm di altezza dei modelli *Full Tower* e *Mid Tower*, fino ai circa 40 cm dei modelli *Mini Tower*.

Le parti di un computer possono essere anche montate su una **singola scheda (blade)** per essere inserite in una struttura meccanica di sostegno (**rack**) che fornisce l'alimentazione elettrica, il raffreddamento e il collegamento in rete (**Figura 2**). La comunicazione tra le schede è assicurata da un **backplane**, un gruppo di connettori elettrici in cui la disposizione parallela dei pin costituisce il bus.

## 4 L'alimentatore

L'alimentatore (**Figura 3**) preleva l'energia dalla rete elettrica, a una tensione di 220 V alternata, e la converte in tensione continua a 24 V. I cavi elettrici che escono dall'alimentatore trasportano la corrente alla scheda madre e si collegano a essa tramite connettori speciali. Lo standard **ATX** (*Advanced Technology Extended*), mostrato nella **Figura 4**, stabilisce le caratteristiche dell'alimentatore.



↑ Fig. 3 Alimentatore con il fascio di cavi elettrici che vanno collegati alla scheda madre e il cavo per la presa di corrente a 220 V.



↑ Fig. 4 Connettore ATX a 24 pin collegato alla scheda madre dall'alimentatore.

### Pit Stop

- 3 Quali sono le componenti del case?
- 4 Come viene trasformata l'energia elettrica dall'alimentatore?

## 5 La scheda madre

La scheda madre (*motherboard*) è l'elemento che ospita tutti i componenti: CPU e chipset, la circuiteria elettronica, i bus di espansione e le interfacce di collegamento verso l'esterno del sistema (**Figura 5**).



← Fig. 5 La scheda madre.

È un circuito stampato ricavato da strati sovrapposti di rame isolati da vetrofite (un composto di fibra di vetro e resina epossidica).

Sul rame sono ricavate le piste che collegano i componenti, sulle quali passano i segnali elettrici a una frequenza molto elevata, dell'ordine dei gigahertz. Questi circuiti devono essere progettati con attenzione per evitare interferenze elettromagnetiche con altri conduttori.

I circuiti integrati, i connettori per il montaggio di schede e gli «zoccoli» (*socket*) su cui installare i chip, sono saldati direttamente sul circuito stampato.

Le motherboard possono avere diversi fattori di forma. Uno dei più diffusi è il formato **ATX** (Figura 4), le cui dimensioni sono 305 × 244 mm.

Nei prossimi paragrafi analizziamo più nel dettaglio due componenti fondamentali della scheda madre: la CPU e il chipset.

### Pit Stop

- 5 Da quali componenti è costituita la scheda madre?

## 6 La CPU

L'elemento centrale collocato sulla scheda madre è la CPU; essa è inserita nello zoccolo chiamato **ZIF** (*Zero Insertion Force*, «a forza di inserimento nulla») per il modo dolce con cui la CPU vi viene alloggiata (Figura 6).



Fig. 6 Posizionamento della CPU.

### Pit Stop

- 6 Spiega come può essere dissipato il calore dalla CPU.

Tutti i processori dispongono di un **dissipatore di calore** simile a quello presente nel raffreddamento ad aria dei ciclomotori (Figura 7a). È costituito da alette in alluminio o rame, attraverso le quali passa il flusso d'aria generato dalla ventola, utile ad asportare il calore. In alcuni casi, l'intensa attività della CPU produce una quantità di calore tale che il dissipatore non è in grado di disperderla efficacemente. Per preservare l'integrità di tutti i componenti si può ricorrere anche a un raffreddamento a liquido (Figura 7b), più efficiente di quello ad aria, proprio come si fa per i motori termici delle auto. In questo caso, una pompa fa circolare del liquido che asporta il calore dai componenti e lo trasporta nel radiatore, rilasciandolo nell'ambiente.



Fig. 7a Ventola di raffreddamento della CPU.



Fig. 7b Sistema di raffreddamento con liquido refrigerante.

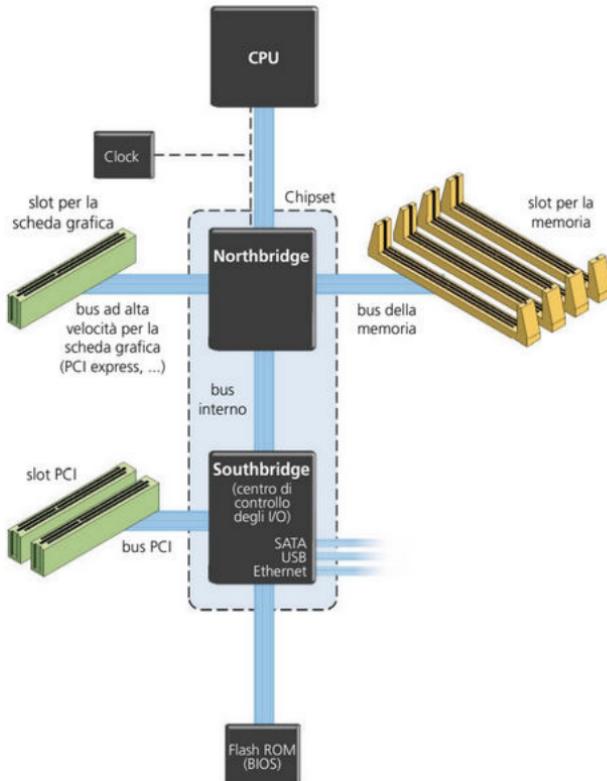
## 7 Il chipset

La CPU è supportata dal **chipset**, un insieme di circuiti integrati che servono al processore per comunicare attraverso i bus con tutti i dispositivi presenti sulla scheda madre. Un chipset è spesso progettato per la specifica famiglia di processori di cui fa parte ed è composto da due parti (come mostra la [Figura 8](#)):

- il **North Bridge**, che riceve il flusso dei dati direttamente dalla CPU e distribuisce il traffico verso i **dispositivi veloci**, tra cui le memorie RAM, le schede di grafiche e la scheda di rete. Per lo scambio dei dati si serve dei principali bus di espansione (PCI e AGP) e delle interfacce ATA e SATA. È l'elemento più critico del chipset e il suo buon funzionamento è cruciale per la stabilità e la velocità della macchina;
- il **South Bridge**, che gestisce tutte le **interfacce a bassa velocità**: è connesso direttamente al North Bridge e collega le porte seriali e parallele, le porte USB e i connettori per le schede di I/O aggiuntive.

### Pit Stop

- Che cos'è il North Bridge?
- Che cos'è il South Bridge?



◀ Fig. 8 Schema logico di una scheda madre.

## 8 La connessione alle periferiche interne

Il **bus di espansione** permette di collegare alla scheda madre **altri dispositivi interni al PC**. Negli slot («alloggiamenti») si possono inserire schede di vario genere, tra cui: schede grafiche e audio, schede di Ingresso/Uscita per l'acquisizione dei dati dall'esterno, ecc. Il bus di espansione si è trasformato seguendo l'evoluzione dei PC, dai primi standard fino a quelli più recenti. Ci sono diversi tipi di bus di espansione, tra cui:

- **PCI (Peripheral Component Interconnect)**, che permette l'interfacciamento delle periferiche con il chipset. Sul bus PCI possono essere collegate schede per l'interfaccia di rete, modem, schede audio e video. Il bus è realizzato da un certo numero di piste compatte messe in **parallelo**;
- **PCIE (PCI Express)**: offre una banda di trasmissione superiore a PCI. PCI Express è un bus **seriale**, mentre il PCI è un bus parallelo;
- **SATA (Serial Advanced Technology Attachment)**: è un'interfaccia standard generalmente utilizzata per il collegamento di dispositivi di memorizzazione, quali hard disk e unità ottiche (**Figura 9a**). La versione **ESATA (External SATA)** permette di collegare dispositivi di memorizzazione esterni al case (**Figura 9b**). Si tratta di un bus seriale in cui l'informazione transita sequenzialmente su cavi separati.



↑ Fig. 9a I cavi SATA, che hanno 15 pin, servono a collegare gli hard disk, le unità ottiche o qualsiasi dispositivo che abbia un connettore di questo tipo.



↑ Fig. 9b Il cavo ESATA serve per collegare dispositivi esterni al case.

## 9 La connessione alle periferiche esterne

Le interfacce collegano la scheda madre ai dispositivi esterni, come tastiere, mouse e le periferiche che dispongono di una connessione USB. I diversi tipi di interfaccia (**Figura 10**) includono:

- **USB (Universal Serial Bus – «bus seriale universale»)**: è stata progettata per connettere più periferiche usando una sola interfaccia e un solo tipo di connettore e consentire il collegamento dei dispositivi «a caldo», cioè senza dover spegnere il computer (*hot swap*). Il sistema USB è asimmetrico, con un gestore al quale sono collegate – tramite dei concentratori (hub) – fino a 127 periferiche, e offre l'alimentazione integrata;
- **SVGA (Super VGA – Video Graphics Array) o UVGA (Ultra VGA)**: sono connettori standardizzati per audio e video;
- connettore **RJ45** per la rete **Ethernet**;
- **Wi-Fi** per il collegamento di una rete locale via radio (**WLAN –Wireless Local Area Network**);

### Pit Stop

- 9** A cosa serve il bus di espansione?  
**10** A che cosa servono le interfacce?

- **Bluetooth** per reti senza filo di tipo personale (**WPAN – Wireless Personal Area Network**) (Figura 11).



↑ Fig. 10 Il retro di una scheda madre di un desktop con le porte per il collegamento delle periferiche esterne: HDMI, USB, Ethernet, DVI, audio.



↑ Fig. 11 Laptop collegato alle cuffie con Bluetooth.

## 10 Le memorie presenti sulla scheda madre

Sulla scheda madre sono presenti la memoria centrale DRAM e la memoria CMOS RAM.

- **La memoria centrale DRAM**

Le schede DIMM, con i chip di memoria, vengono inserite in appositi alloggiamenti (slot) presenti sulla scheda madre (Figura 12).

- **La memoria CMOS-RAM**

È una memoria RAM che, opportunamente alimentata da una batteria, è capace di mantenere i parametri di configurazione del sistema anche a computer spento.

Questo circuito integrato, realizzato con tecnologia CMOS, contiene un orologio-calendario (*clock-calendar*) e uno spazio di qualche decina di byte di RAM, utilizzato per conservare i dati variabili della configurazione del PC, come il tipo di disco, la scheda video, il coprocessore, la quantità di memoria interna, ecc. È collegato a una batteria tampone (Figura 13) che lo alimenta anche quando si spegne il PC. È gestito come una periferica di I/O. La CMOS-RAM è mappata con indirizzi compresi tra 70h e 7Fh (dove h significa «esadecimale»).

### notabene

**CMOS** (Complementary MOS) è la tecnologia dominante per la realizzazione dei circuiti integrati. Offre vantaggi quali il basso consumo energetico, le dimensioni ridotte e i bassi costi di produzione.



↑ Fig. 12 DIMM con memoria RAM installata nello slot della scheda madre del computer.



↑ Fig. 13 La batteria che alimenta la CMOS-RAM dissipava una potenza di pochi nanowatt (nW) e si scarica in alcuni anni.

### Pit Stop

- 11 Quali sono le memorie presenti sulla scheda madre?

## 11 Il BIOS e l'avvio del sistema

Il **BIOS** (*Basic Input-Output System*) fornisce i primi comandi per l'avvio (**boot** o **bootstrap**) della macchina e provvede al caricamento del sistema operativo. È un **firmware** (un programma inserito direttamente in un chip) memorizzato in una ROM o, più frequentemente, in una memoria flash.

Nelle schede madri più recenti, in alternativa al BIOS, è installato il firmware **UEFI** (*Unified Extensible Firmware Interface*) che prevede funzioni aggiuntive rispetto a quelle già presenti nel BIOS e, soprattutto, maggiore attenzione alle procedure di sicurezza, tra cui la possibilità di abilitare:

### Pit Stop

**12 Che cos'è il BIOS?**

**13 Che cos'è il UEFI?**

- la crittografia (con l'applicazione *BitLocker Drive Encryption*), che è utilizzata per proteggere i dischi rigidi e impedire l'accesso ai dati da parte di malintenzionati;
- l'avvio protetto (*secure boot*), che garantisce che i dispositivi avviano solo un sistema attendibile: all'avvio è controllata la validità (la «firma») dei driver del firmware UEFI e del sistema operativo installato.

In generale, le fasi di avvio di un PC possono essere così descritte:

- **reset hardware.** Quando viene data alimentazione elettrica alla scheda madre, la CPU salta all'indirizzo FFFF0h (o FFFFFFF0h) in cui è «cablata» un'istruzione di salto (*jump*) all'*entry point* nel BIOS;
- **POST (Power-On Self Test).** Esegue il controllo del sistema. Dalla CMOS-RAM è letta la configurazione del sistema, impostata dal fornitore oppure modificata dall'utente. È eseguito il **test dell'hardware** (RAM, dischi, periferiche). Se il test ha esito negativo, il sistema si blocca e un messaggio sul monitor associato a un segnale acustico segnala l'errore;
- **caricamento** nella RAM del **codice per i servizi di base**;
- **ricerca del boot sector:** il BIOS carica nella RAM il Sistema Operativo, che si trova sulla memoria di massa. In base a un ordine prestabilito e configurabile, viene cercato un settore in cui è contenuto il codice iniziale per l'avvio del sistema operativo (Windows, Linux o altro);
- **caricamento del sistema operativo e avvio.** Viene caricato il kernel del sistema operativo. Questa fase esaurisce la preparazione all'avvio del sistema e il controllo passa al sistema operativo: il sistema è partito!



↑ Fig. 14 Il BIOS è contenuto in una ROM presente sulla scheda madre. L'accesso alle impostazioni del BIOS avviene premendo un tasto specifico durante la fase di POST, appena dopo il reset.

Il termine *bootstrap* si riferisce al modo di dire inglese «pull yourself up by your bootstraps», che tradotto letteralmente vuol dire «sollevarsi da terra da solo prendendosi per le stringhe degli stivali». Con questa azione paradossale si vuole intendere la capacità di uscire da una situazione difficile contando solo sulle proprie forze. In pratica con il processo di bootstrap, durante la fase di avvio, un piccolo pezzo di codice contenuto nel BIOS viene impiegato per caricare il codice più complesso del sistema operativo, finché la macchina è pronta per l'uso.

Il BIOS è contenuto in una memoria ROM (Figura 14) e si può trovare sulle schede madri in varie tecnologie: **PROM** (*Programmable Read-Only Memory*), EEPROM (*Electrically Erasable Programmable Read-Only Memory*), flash memory.

## Comprendi con l'analogia

### Il BIOS opera come la parte inconscia del nostro cervello al risveglio

Dopo una passeggiata impegnativa in montagna, nella quale mi ero stancato molto, rientrato a casa a tarda ora ero stato colto da un sonno pesante ed ero crollato come un sasso. Quando suonò la sveglia, non riuscivo a capire dove mi trovassi. La testa mi pulsava. Cercavo di aprire gli occhi e mi sentivo le gambe pesanti. Dopo qualche sforzo, riuscii a sollevarmi dal letto. Il contatto freddo dei piedi scalzi con la ceramica delle piastrelle mi scosse e mi trascinai in cucina. Fu solo dopo un caffè nero, bollente, che ripresi contatto con la realtà degli impegni che mi aspettavano e che erano spariti dalla mia memoria a breve termine e del ritardo che avevo già accumulato...

Al risveglio la parte inconscia del mio cervello (il BIOS) aveva verificato i miei parametri vitali: memoria, occhi, gambe (ingressi/uscite). Solo in un secondo momento era intervenuta la parte consci (il sistema operativo) che aveva caricato la memoria di lavoro (RAM) e aveva elaborato quello che dovevo fare.

## 12 Le schede grafiche e la GPU

Le schede grafiche elaborano le immagini utilizzando una propria memoria, sollevando così la CPU da calcoli pesanti.

Le schede grafiche (**Figura 15**) contengono una **GPU (Graphics Processing Unit)**, cioè un processore capace di accelerare il processo di creazione e manipolazione delle immagini.

La GPU è composta da molti core, specializzati in attività di calcolo. La suddivisione del carico di lavoro tra i core permette di ottenere prestazioni molto elevate. Anche se le specialità delle GPU rimangono le immagini 3D e la grafica dei videogiochi, questi processori sono la base per molte applicazioni che richiedono carichi di lavoro impegnativi, come succede per gli algoritmi dell'Intelligenza Artificiale e il *data mining*, cioè l'analisi dei dati, e per le criptovalute.



◀ Fig. 15 Montaggio nel caso di un computer da gioco di una scheda grafica con le ventole per il raffreddamento.

Le schede grafiche possono essere inserite nel bus di espansione all'interno del desktop. Esse permettono di ottenere prestazioni eccellenti, ma a un costo elevato. Una scelta più economica è quella proposta da alcuni produttori che integrano la GPU nella CPU: gli ingombri diminuiscono e aumenta l'efficienza energetica. Tuttavia, la GPU condivide in tal caso con la CPU l'utilizzo della memoria e questo ne riduce le prestazioni.

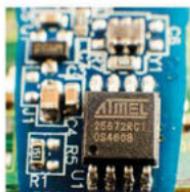


Fig. 16 ATMEL 8-bit, un microcontrollore della famiglia RISC.

## 13 Il microcontrollore: un elaboratore in un chip

Un microcontrollore (**μC** o **MCU - Micro Controller Unit**) è un computer completo di unità di elaborazione, memoria e interfacce di I/O, integrate in un unico chip. Introdotto fin dagli anni Ottanta del secolo scorso, il microcontrollore ha rappresentato spesso un'alternativa al microprocessore che, per quanto potente, ha tuttavia bisogno dei chip di memorie e di interfaccia. La sua versatilità lo ha portato al successo e molti produttori propongono famiglie di microcontrollori basate su microprocessori che appartengono sia alla famiglia CISC sia RISC (Figura 16).

La struttura di base di un microcontrollore comprende:

- CPU a 8, 16 o 32 bit;
- memoria volatile (RAM) e permanente (ROM, EEPROM, memoria flash) per l'archiviazione dei dati e dei programmi;
- porte di ingresso/uscita parallele e seriali, per interfacciarsi al mondo esterno e pilotare le periferiche;
- timer o contatori per la generazione di impulsi e la misurazione di frequenze;
- ADC (convertitore da analogico a digitale), utilizzato per convertire i segnali analogici in segnali digitali;
- DAC (convertitore da digitale ad analogico): questo convertitore esegue funzioni opposte rispetto all'ADC;
- porte aggiuntive per funzioni speciali e dedicate.

### Pit Stop

- 14 Quali fattori hanno determinato il successo dei microprocessori?

Il successo dei microcontrollori è dovuto a vari fattori, tra cui i bassi costi e consumi e la facilità di programmazione.

Sono diffusi ovunque: nei sistemi robotici, in quelli di controllo industriale, nelle auto, nei giochi e nei sistemi IoT (*Internet of Things*).

## 14 Il SoC: una scheda madre in un chip

Immaginate di avere una scheda madre con tutte le più potenti schede di calcolo e di grafica e di ridurne le dimensioni a quelle di un chip. Avete nelle mani un **SoC (System-on-a-Chip)** (Figura 17).



Fig. 17 SoC Snapdragon di Qualcomm. Molti sono i produttori che si contendono il mercato con dispositivi che integrano sempre più funzioni che comprendono la grafica avanzata, la connettività WiFi e 5G e le unità di elaborazione legate all'Intelligenza Artificiale.

Un SoC è un chip che racchiude un intero sistema di elaborazione, ha piccole dimensioni (circa di 2 cm × 2 cm) e consuma circa 2 W. Comprende uno o più processori a basso consumo, coprocessori matematici e GPU, che aggiungono

funzionalità, quali l'aritmetica in virgola mobile, la grafica e la crittografia, liberando il processore centrale da pesanti elaborazioni e calcoli.

I SoC sono conosciuti soprattutto per il loro utilizzo nel campo della telefonia mobile e nei microcontrolleri *embedded* («integriti»).

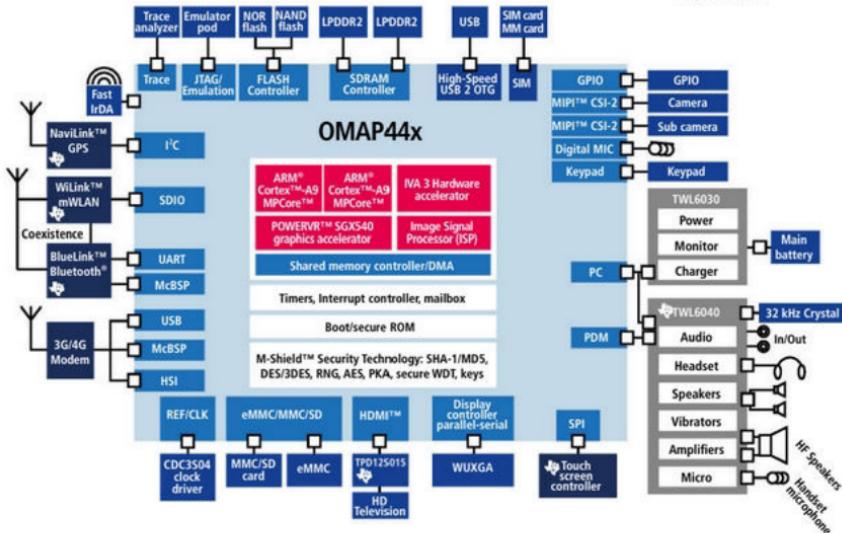
La **Figura 18** mostra un esempio che contiene i blocchi principali che sono alla base del SoC, cioè:

- il **processore**: è il cuore del SoC, solitamente costituito da più core;
- il **DSP (Digital Signal Processor)**: esegue operazioni di elaborazione del segnale;
- la **memoria centrale**: RAM (SRAM e DRAM), ROM;
- la **memoria secondaria**: flash e EEPROM;
- l'**Encoder/Decoder** («codificatore/decodificatore»): utilizzato per la codifica delle informazioni;
- la **GPU (Graphical Processing Unit)** per la grafica avanzata;
- il **DMA (Direct Memory Access)**: per il trasferimento di dati tra la memoria e i dispositivi esterni;
- le **interfacce di comunicazione**, come USB, HDMI, Wi-Fi, *Bluetooth* e modem integrati per 5G/LTE. Per la trasmissione seriale sono usati i moduli **USART (Universal Synchronous-Asynchronous Receiver/Transmitter)**, «ricevitori/trasmettitori universali sincroni-asincroni» e **SPI (Serial Peripheral Interface)**, «interfaccia periferica seriale», per i sistemi mobili sono usati i modem 5G;
- i **convertitori analogici/digitali e i convertitori digitali/analogici**.

### Pit Stop

#### 15 Che cos'è un SoC?

↓ Fig. 18 Un esempio di SoC: il sistema integrato OMAP44x. In rosso, i blocchi critici che influenzano le prestazioni.



La gestione di un hardware così complesso è affidata a **driver software**, che permettono il controllo e l'interazione tra i vari blocchi che compongono il SoC. Oltre al software, è essenziale dotare questi dispositivi di sistemi di sicurezza, in particolare sui dispositivi mobili, per garantire la privacy e la riservatezza dei dati. La consapevolezza che un'applicazione software è potenzialmente più vulnerabile agli attacchi rispetto a soluzioni hardware ha portato a implementare dispositivi hardware che sono fortemente legati agli algoritmi software di crittografia. All'interno del chip sono stati inseriti componenti dedicati a questo scopo e segregati dal resto. Essi sono in grado di generare chiavi di crittografia univoci per ogni tipo di operazione e non danno la possibilità che malware di qualsiasi genere possano accedere alle chiavi memorizzate.

### Pit Stop

- 16 Perché è necessario proteggere i SoC da attacchi esterni?

### I SoC in una visione allargata

È molto probabile che dietro le quinte del dispositivo mobile che stiamo utilizzando (smartphone o tablet o laptop) o indossando (smartwatch) in questo momento, sia nascosto un SoC che fa da ricettore ed elaboratore dei segnali che riceve dai sensori a esso collegati.

I SoC con le loro dimensioni ridotte, i bassi consumi e la possibilità di essere sempre connessi, hanno cambiato il modo di interpretare l'utilizzo dei dispositivi di cui sono il cuore, con ripercussioni sulla nostra vita.

L'introduzione dei sistemi wearable ha creato un ecosistema rappresentato dal triangolo di **Figura 19**. In un vertice c'è lo smartphone, negli altri due ci sono due sistemi wearable: smartwatch e auricolari wireless (earbuds). La forza che li unisce si chiama connettività, che permette scambio di informazioni e arricchimento reciproci. Ai lati del triangolo sono rappresentati altri dispositivi:

- i visori permettono di immergersi nella realtà aumentata e virtuale (AR/VR);
- l'abbigliamento, le scarpe e i tracker sono utilizzati per lo sport e il tempo libero;
- i sensori presenti su diversi dispositivi permettono la realizzazione dei servizi di benessere personale, il controllo a distanza dei pazienti e degli animali domestici;
- la connettività permette l'interazione, in sicurezza e a mani libere, con altri sistemi (videogiochi, auto, sistemi di automazione, ecc.);
- i servizi di Intelligenza Artificiale, presenti in diversi oggetti, sono utili per la formazione e la didattica.



Fig. 19 Il triangolo formato dallo smartphone, dallo smartwatch e dagli earbuds, con ai lati gli altri dispositivi digitali, rappresenta in modo schematico un nuovo ecosistema comunicativo.

## 15 SBC: un computer in una scheda

Un **SBC** (**S**ingle **B**oard **C**omputer, «computer a scheda singola») è a tutti gli effetti un computer completo di processore, memoria, interfacce con le periferiche e altre funzionalità, ospitato su un'unica scheda.

In questi ultimi anni stiamo assistendo ad un crescente interesse verso i *Single Board Computer* che rappresentano la soluzione ideale per molte applicazioni specializzate, dove i consumi, l'affidabilità e il costo sono determinanti. Le dimensioni di un SBC sono simili, o inferiori, a quelle di una carta di credito e adottano fattori di forma standardizzati e ridotti.

Gli SBC utilizzano microprocessori RISC, memorie RAM e schede di memoria a stato solido SD (**S**ecure **D**igital) per la memorizzazione dei dati e dei programmi. Il basso consumo è una caratteristica importante degli SBC, tuttavia nessuno di essi presenta sistemi di raffreddamento, per cui occorre prestare attenzione ai consumi e al calore eccessivi durante il loro funzionamento.

Uno dei punti di forza è la facilità di interfacciarsi ad hardware esterni. A differenza dei PC, i single board non hanno bus di espansione in cui inserire le schede aggiuntive, ma al loro posto dispongono di **connettori specifici**, costituiti da un insieme di pin tramite i quali è possibile controllare degli attuatori e ricevere segnali dai sensori o comunicare con altri sistemi.

Gli SBC si adattano agli usi più diversi, con modelli che hanno prestazioni e fattori di forma differenti.

In questa sede ci limitiamo a tracciare le caratteristiche di due tra i sistemi SBC più utilizzati: **Raspberry Pi** e **Arduino**. Entrambi i sistemi dispongono di numerose schede aggiuntive (*shield*) per controllare sensori e attuatori e hanno in comune il fatto di essere **sistemi aperti**.

### Raspberry Pi

Raspberry Pi è un computer completo, basato su un *System on a Chip* (Figura 20a). È dotato di sistema operativo e porte di vario genere (USB, Ethernet, Wi-Fi). Può essere usato con diversi linguaggi di programmazione, che lo rendono adatto a moltissime applicazioni. Il **software** di Raspberry Pi è **open source**, mentre gli schemi hardware vengono rilasciati come documentazione, ma la scheda non è un hardware aperto. È stato lanciato nel 2012 da Raspberry Pi Foundation come progetto in ambito educativo.

### Pit Stop

- 17 Che cos'è un SBC?
- 18 Quali sono i vantaggi di un SBC?



Fig. 20a Un esempio di SBC: Raspberry Pi Pico;



Fig. 20b Tre esempi di MCU: Arduino Uno, Arduino Mega e Arduino Nano.

### Arduino

Arduino è una scheda, basata su microcontrollore (Figura 20b) che può essere programmata in un linguaggio molto simile al linguaggio C, ma non ha un sistema operativo e gestisce operazioni semplici. Arduino è una piattaforma open source sia software sia hardware: gli schemi delle schede sono pubblicati sotto una licenza Creative Commons e chiunque può realizzare la propria versione hardware, migliorando o estendendo quella originale. È stato sviluppato nel 2005 dall'Interaction Design Institute di Ivrea come strumento didattico e professionale.

# Laboratorio

## Montaggio delle parti di un PC desktop



**Galleria**  
Montaggio  
del computer

### ATTIVITÀ 1

#### Montaggio passo per passo delle parti di un PC desktop

##### Scenario

Disponiamo delle parti di un PC desktop:

- case;
- alimentatore;
- motherboard;
- RAM;
- CPU;
- ventola di raffreddamento;
- schede aggiuntive (di rete, grafica, ecc.);
- periferiche: dischi magnetici e ottici;
- cavi di alimentazione;
- cavi di bus Parallel ATA, Serial ATA;
- cavi per il cablaggio esterno: cavi per monitor, USB, LAN Ethernet.

##### Consegna

Montare passo per passo le parti, ottenendo il PC desktop.

##### Svolgimento



↑ Passo 1 Inserimento dell'alimentatore nel case.



↑ Passo 2 Preparazione della scheda madre.



↑ Passo 3 Inserimento della RAM.



↑ Passo 4 Inserimento della CPU.



↑ Passo 5 Inserimento della ventola di raffreddamento.



↑ Passo 6 Inserimento e fissaggio della motherboard nel case.



↑ Passo 7 Inserimento delle schede aggiuntive: di rete, grafica, ecc.



↑ Passo 8 Inserimento delle periferiche: dischi magnetici e ottici.



↑ Passo 9 Inserimento dei cavi di alimentazione nella motherboard e nei dischi.



↑ Passo 10 Collegamento dei cavi di bus Parallel ATA, Serial ATA.

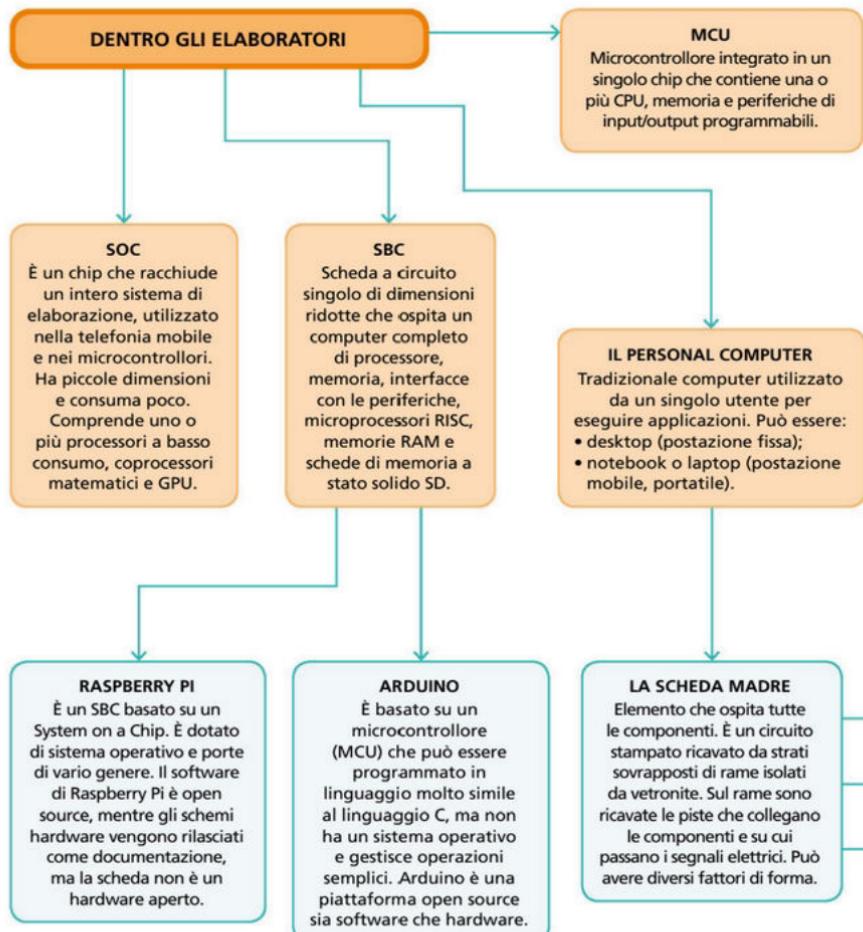


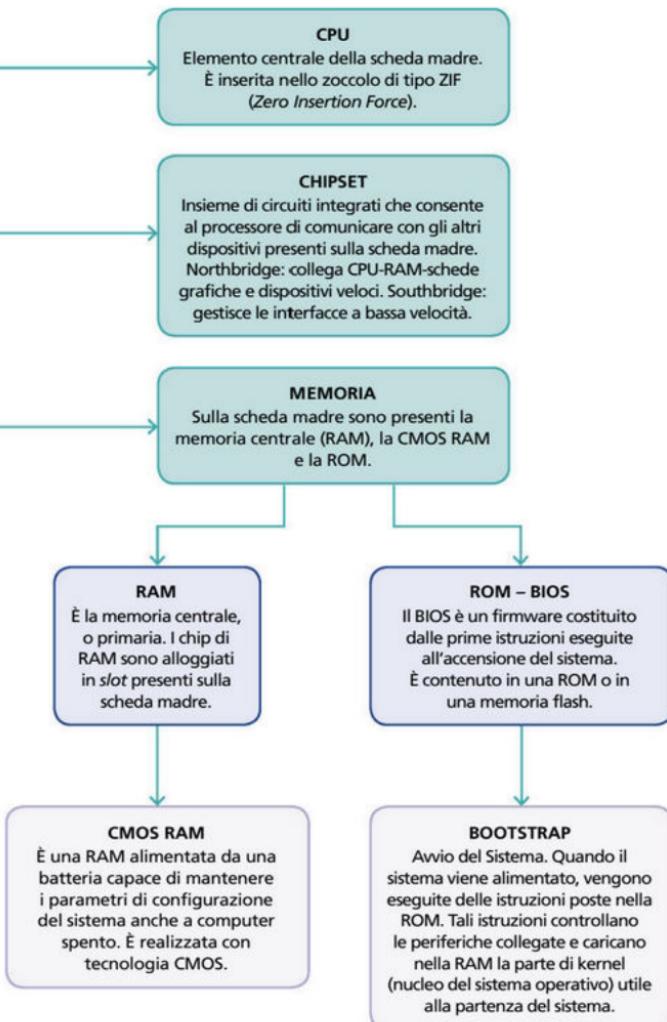
↑ Passo 11 Cablaggio esterno: cavi per monitor, USB, LAN Ethernet.



↑ Passo 12 Completato!

# SINTESI





# Verifica delle CONOSCENZE

Unità 6



## Vero o Falso?

- 1 Il desktop è un computer usato in mobilità.
- 2 Il chipset della scheda madre è una scheda grafica veloce.
- 3 Il BIOS è un firmware.
- 4 I cavi di alimentazione forniscono energia a tutto il sistema.
- 5 Il case è contenuto nella scheda madre.
- 6 La scheda madre è un circuito stampato ricavato da strati sovrapposti di rame.
- 7 Il socket è un modulo di RAM.
- 8 La ROM è una parte del chipset della scheda madre.
- 9 AGP è una scheda grafica.
- 10 La CPU deve essere posta nell'apposito socket premendo con forza per evitare che i contatti non funzionino bene.
- 11 PCI Express è un bus seriale.
- 12 I SoC sono sistemi su un unico chip.
- 13 Arduino è un SBC.
- 14 Raspberry Pi ha un'architettura RISC.
- 15 Arduino è un microcontrollore.
- 16 La CMOS-RAM è alimentata con batteria anche a computer spento.
- 17 Il BIOS è una memoria a sola lettura.
- 18 POST è una fase dell'avvio del computer.
- 19 Al reset hardware la CPU esegue l'istruzione posta all'indirizzo FFFF0h.
- 20 POST è la ricerca del boot sector.

V

F

V

F

V

F

V

F

V

F

V

F

V

F

V

F



## Rispondi ai seguenti quesiti indicando l'unica risposta esatta.

- 21 Un desktop è:
  - a un notebook
  - b un laptop
  - c un personal computer
  - d un tablet
- 22 L'alimentatore:
  - a è costituito da un supporto isolante
  - b è un connettore utile per le schede
  - c è una periferica di output
  - d fornisce energia elettrica
- 23 La scheda madre:
  - a mette in comunicazione vari dispositivi
  - b è in fibra ottica
  - c contiene solo CPU e RAM
  - d nessuna delle precedenti risposte è vera
- 24 Un esempio di bus seriale è:
  - a PCI Express
  - b MCU
  - c ADC
  - d chipset
- 25 Lo standard USB:
  - a permette la comunicazione tra PC e molte periferiche
  - b è un'interfaccia parallela a velocità molto alta
  - c è usato per reti locali
  - d non consente il collegamento dei dispositivi senza spegnere il computer
- 26 La ROM:
  - a contiene il sistema operativo
  - b è contenuta nel BIOS
  - c è una flash memory
  - d è una memoria in sola scrittura
- 27 Che cos'è la scheda madre?
- 28 Che cosa può essere collegato alla scheda madre?
- 29 Che cosa accomuna l'architettura di un SoC con la scheda madre?
- 30 Quali sono le principali tipologie di computer?
- 31 Che cos'è il case?
- 32 Che cos'è un SBC?
- 33 A che cosa serve l'alimentatore?
- 34 Perché è necessario un sistema di ventilazione all'interno del case di un computer?
- 35 Quali sono i principali connettori di alimentazione della scheda madre?
- 36 Che cos'è il chipset della scheda madre?
- 37 Come deve essere montata la CPU sulla scheda madre?
- 38 Che cosa significa CMOS per la scheda madre?
- 39 Qual è la differenza tra BIOS e CMOS-RAM?
- 40 Qual è la principale funzione del BIOS?
- 41 Quali sono le fasi di avvio di un computer?
- 42 A che cosa serve la ROM?

## Domande per la prova orale

- 27 Che cos'è la scheda madre?
- 28 Che cosa può essere collegato alla scheda madre?
- 29 Che cosa accomuna l'architettura di un SoC con la scheda madre?
- 30 Quali sono le principali tipologie di computer?
- 31 Che cos'è il case?
- 32 Che cos'è un SBC?
- 33 A che cosa serve l'alimentatore?
- 34 Perché è necessario un sistema di ventilazione all'interno del case di un computer?

### Esegui i compiti seguenti.

43 Descrivi il modello logico della scheda madre.

44 Completa la seguente tabella.

Dispositivo	Descrizione	Vantaggi
Desktop		
Laptop		
Tablet		
Telefono		

45 Completa la seguente tabella.

Interfaccia	Descrizione	Vantaggi
USB		
Ethernet		
FireWire		

46 Fai una ricerca sul web e sintetizza in una tabella i fattori che devono essere presi in considerazione per scegliere la scheda madre.

47 Fai una ricerca sul web e completa la tabella con le informazioni relative ai fattori di forma della scheda madre.

Tipo di scheda	Dimensioni	Memoria RAM	PCI express	GPU
ATX (standard)				
Micro-ATX				
Mini-ATX				

48 Riassumi in una tabella, le caratteristiche di Raspberry Pi e Arduino, evidenziando i punti in comune e le differenze.

49 Fai una ricerca sul web e riassumi le principali differenze tra un SoC e un normale processore.



Esercizi in più



## Inside the Personal Computer

### Abstract

There are several types of Personal Computers, characterized by different dimensions, computing performances, and usage: workstation, desktop, laptop, tablet, ...  
The motherboard (or mainboard) is the main component inside a PC. The motherboard is

a printed circuit board. All the other internal components, CPU, RAM, ROM, disks, video card, I/O interfaces, network card, buses, cooling fan, power supplier are directly mounted on the mainboard or connected to it.

### Questions

#### Answer all questions

- 1 What computer types do you know?
- 2 What's the computer case?
- 3 What's the motherboard?
- 4 Can you list the main stages of bootstrap?

#### Choose the correct answer



- 5 The power supplier:
  - [a] is an auxiliary 4/6 pins connector
  - [b] is a printed circuit
  - [c] supplies electric power to the computer system
  - [d] has a fan cooler

#### 6 CMOS-RAM:

- [a] is a read only memory
- [b] is a non volatile memory
- [c] is used to execute a POST
- [d] is a high power consumption device

#### 7 The video card:

- [a] is used to read a program
- [b] is mounted on the CPU socket
- [c] contains a calendar-clock
- [d] manages the output images of a display

#### 8 SBC is:

- [a] a computer with all components integrated on a single circuit board
- [b] the start-up of a system
- [c] a type of motherboard
- [d] a firmware

### Crosswords

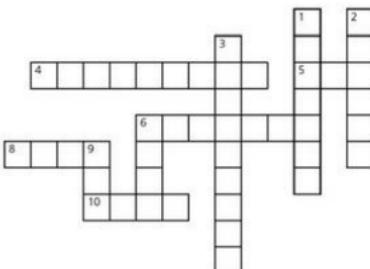
#### Across

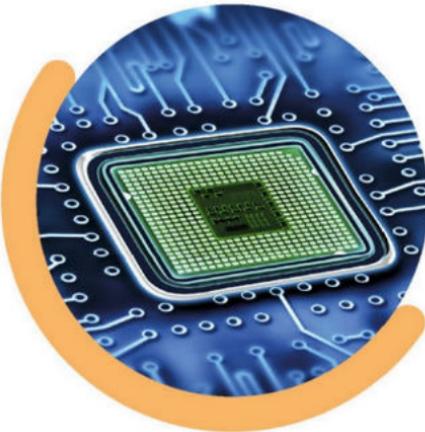
- 4 Start-up of a system
- 5 A complete computer built on a single circuit board
- 6 Allows the CPU to communicate with other components attached to the motherboard
- 8 Can be found in the ROM
- 10 Contains the hardware of a PC

#### Down

- 1 Personal Computer used at a fixed location
- 2 Installation site of a CP
- 3 An example of SBC
- 6 Has the configuration parameters of a system
- 9 An integrated circuit that contains a system on

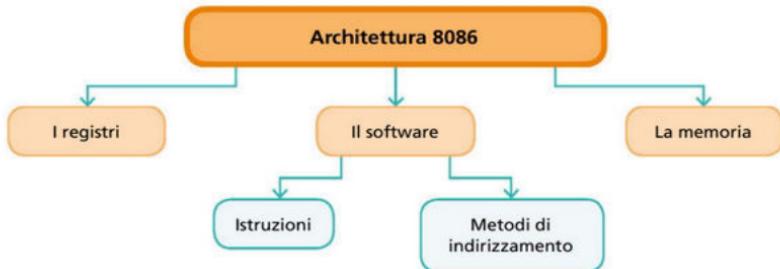
a single chip





# Unità 7

## L'architettura del microprocessore 8086



### Visione d'insieme

- Architettura del microprocessore 8086.
- Registri e gestione della memoria.
- Set di istruzioni e metodi di indirizzamento.



Fig. 1 Intel Core, un processore multi-core, è la pietra d'angolo su cui poggia la maggior parte dei Personal Computer desktop e laptop attuali. X64 è la famiglia che estende il set di istruzioni a 64 bit.

## 1 Introduzione

Il primo microprocessore che vogliamo studiare è 8086, il capostipite della famiglia x86 progettata da Intel (Figure 1 e 2). Ha un'architettura CISC con un set di istruzioni (**ISA - Instruction Set Architectures**) compatibile con i modelli che sono stati introdotti in seguito e che hanno avuto un'evoluzione, in termini di prestazioni, fino ad arrivare a Core iX. La maggior parte dei Personal Computer desktop e laptop è basata su sistemi della famiglia x86.



Fig. 2 La sede di Intel Corporation a Santa Clara in California.

## 2 Architettura 8086

Le principali caratteristiche dell'architettura dell'8086 (Figura 3) sono:

- **bus dati a 16 bit.** È possibile trasferire 2 byte alla volta in un'unica operazione;
- **bus indirizzi a 20 bit**, in grado di indirizzare 1 megabyte (MB) di memoria, cioè  $2^{20}$  locazioni;
- **14 registri a 16 bit.**

La CPU è divisa in due parti: la **EU (Execution Unit)** preposta all'esecuzione delle istruzioni e la **BIU (Bus Interface Unit)**, responsabile della comunicazione con il bus. La caratteristica più rilevante del sistema è che queste due parti **possono lavorare autonomamente l'una rispetto all'altra**, e quindi possono fare contemporaneamente due cose diverse, in modo indipendente. Mentre la EU esegue un'istruzione (fase di execute), la BIU può cominciare a prelevare dalla memoria l'istruzione successiva. In questo modo è possibile rendere *parallele* (contemporanee) le operazioni di accesso al bus esterno (fetch) e le operazioni di elaborazione. Le due parti comunicano tramite una *coda di prefetch* (una coda di registri gestita con politica FIFO, First In First Out): mentre la BIU pone nella coda le istruzioni da eseguire, la EU le «preleva» dalla coda e le esegue.

È questo un primo esempio di realizzazione della **tecnica del pipeline**. La coda di prefetch ha una dimensione di 6 byte, che è la dimensione massima di un'istruzione assembly 8086.

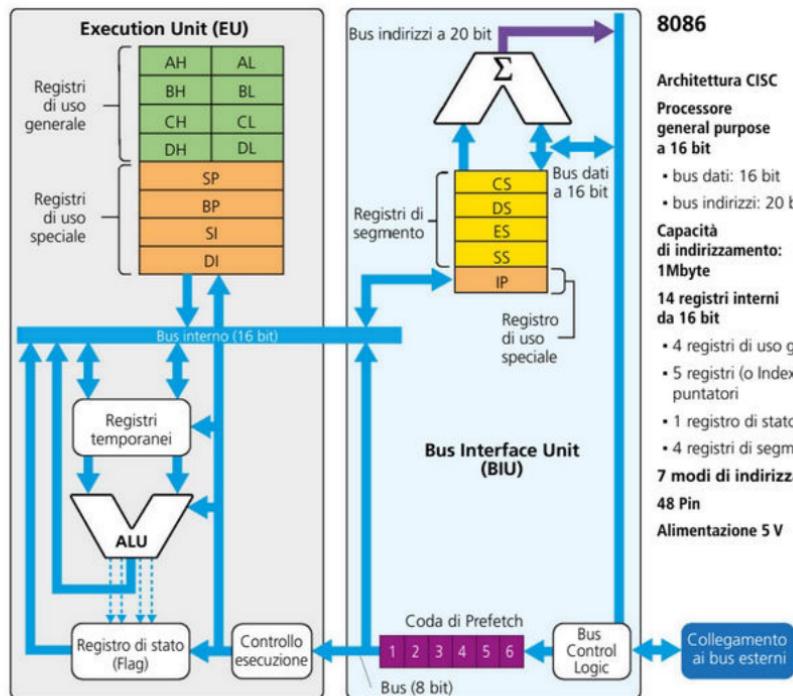
La BIU effettua una lettura anticipata (pre-fetch) delle istruzioni dalla memoria in modo sequenziale e le pone nella coda.

Poiché questa è gestita con politica FIFO, la prima istruzione che entra nella coda sarà la prima a essere prelevata dalla EU per essere eseguita. In questo modo le istruzioni sono anticipatamente caricate ed eseguite nell'ordine corretto.

Questo meccanismo ha uno svantaggio: la lettura delle istruzioni è strettamente sequenziale, per cui in caso di salto (o interruzione) la coda predisposta è inutile e va ricostruita.

### Pit Stop

- 1 A che cosa serve la EU?
- 2 A che cosa serve la BIU?
- 3 Che cosa si intende per politica FIFO?



### Comprendi con l'analogia

La CPU e l'unità di prefetch collaborano così come fanno un muratore e un manovale

Mi chiamo Francesco e sono un muratore. Lavoro in proprio da molti anni e cerco sempre di fare un lavoro di qualità. Una chiave del mio successo è Alfredo, un bravo manovale che collabora con me. Quando salgo su un'impalcatura per costruire un muro, Alfredo mi riempie il secchio di malta di cemento. Mentre procedo alla costruzione, posizionando un mattone sopra l'altro, Alfredo prepara un secondo secchio e me lo porta in modo che io, terminato il primo, passi subito al successivo.

Il giorno in cui Alfredo si è preso un brutto raffreddore e ho dovuto fare a meno di lui, ho capito quanto era preziosa la sua collaborazione: ci metteva più tempo a mescolare il cemento con l'acqua e portarlo sull'impalcatura, che a costruire il muro! La CPU che processa le istruzioni velocemente è proprio come me, perché la velocità di esecuzione è una sua caratteristica essenziale. Alfredo è come la BIU che preleva dalla memoria le istruzioni e le fa trovare pronte alla CPU nella coda di memoria veloce.



### 8086

#### Architettura CISC

#### Processore general purpose a 16 bit

- bus dati: 16 bit
- bus indirizzi: 20 bit

**Capacità di indirizzamento:**  
1Mbyte

**14 registri interni da 16 bit**

- 4 registri di uso generale
- 5 registri (o Index o Offset) puntatori
- 1 registro di stato (Flag)
- 4 registri di segmento

#### 7 modi di indirizzamento

48 Pin

Alimentazione 5 V

↑ Fig. 3 La CPU 8086 è divisa in due blocchi: l'unità di esecuzione (EU) e l'unità di interfaccia con i bus (BIU), collegate da una coda di prefetch. La BIU effettua una lettura anticipata (prefetch) delle istruzioni dalla memoria e le pone nella coda, da dove l'EU le preleva nello stesso ordine di arrivo.

### 3 I registri

La CPU 8086 ha 14 registri tutti di 16 bit, descritti di seguito ([Tabella 1](#)).

Registri di uso generale	
Registro	Descrizione
<b>AX</b>	<b>Accumulation Register</b> (Registro accumulatore). Può essere separato in due registri di 8 bit: <b>AH</b> (parte alta) e <b>AL</b> (parte bassa).
<b>BX</b>	<b>Base Register</b> (Registro base). È utilizzato per contenere la base di un indirizzo relativo. Può essere separato in due registri di 8 bit: <b>BH</b> (parte alta) e <b>BL</b> (parte bassa).
<b>CX</b>	<b>Counter Register</b> (Registro contatore). È utilizzato, per esempio, nell'istruzione LOOP. Può essere separato in due registri di 8 bit: <b>CH</b> (parte alta) e <b>CL</b> (parte bassa).
<b>DX</b>	<b>Data Register</b> (Registro Dati). È utilizzato in istruzioni aritmetiche e di I/O. Può essere separato in due registri di 8 bit: <b>DH</b> (parte alta) e <b>DL</b> (parte bassa).
Registri di uso speciale	
Registri indice e puntatori (Contengono un indirizzo)	
Registro	Descrizione
<b>SI</b>	<b>Source Index.</b> È utilizzato come puntatore nelle istruzioni di manipolazione di stringhe.
<b>DI</b>	<b>Destination Index.</b> È utilizzato come puntatore nelle istruzioni di manipolazione di stringhe.
<b>BP</b>	<b>Base Pointer.</b> È il puntatore alla base dello stack. È utilizzato nelle operazioni sullo stack.
<b>SP</b>	<b>Stack Pointer.</b> È il puntatore alla cima dello stack. È utilizzato nelle operazioni sullo stack.
<b>IP</b>	<b>Instruction Pointer.</b> È il Program Counter.
Registri di segmento	
Sono utilizzati per la segmentazione della memoria, una tecnica per gestire blocchi di memoria contenenti dati o istruzioni, come sarà spiegato nei prossimi paragrafi.	
Registro	Descrizione
<b>CS</b>	<b>Code Segment.</b> Contiene l'indirizzo di inizio del <i>code segment</i> .
<b>DS</b>	<b>Data Segment.</b> Contiene l'indirizzo di inizio del <i>data segment</i> .
<b>ES</b>	<b>Extra Segment.</b> Contiene l'indirizzo di inizio dell' <i>extra segment</i> .
<b>SS</b>	<b>Stack Segment.</b> Contiene l'indirizzo di inizio dello <i>stack segment</i> .
Registro di stato	
Definisce lo stato della CPU	
Registro	Descrizione
<b>SR</b>	<b>Status Register.</b> I 16 bit dello SR contengono 9 flag divisi in due gruppi: 6 bit di stato e 3 di controllo, come è descritto nella <a href="#">Figura 4</a> . Gli altri non sono utilizzati.

→ Tab. 1

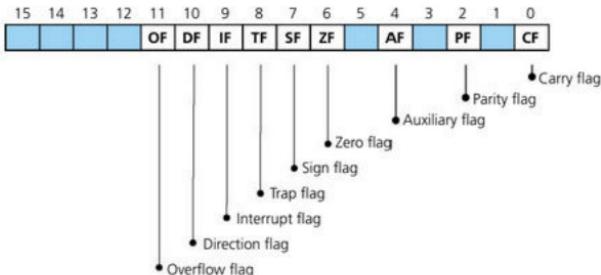


Fig. 4 Bit del registro di stato.

= Bit di stato

= Bit di controllo / = Non utilizzati



Bit	Nome		Descrizione
B0	CF	Carry	1 se c'è riporto o prestito 0 altrimenti
B2	PF	Parity	1 se il risultato (o valore) ha un numero pari di 1 0 altrimenti
B4	AF	Auxiliary	1 riporto tra il bit 3 e il bit 4; è usato per operazioni sui numeri BCD 0 altrimenti
B6	ZF	Zero	1 se il risultato è 0 0 altrimenti
B7	SF	Sign	1 se il numero espresso in complemento a 2 è negativo 0 altrimenti
B8	TF	Trap	1 La CPU sta compiendo un'esecuzione passo passo. Una istruzione per volta. Risultati parziali 0 L'esecuzione si ferma al completamento del programma. Risultati finali
B9	IF	Interrupt (può essere settato da programma)	1 CPU risponderà a un interrupt mascherabile 0 interrupt mascherabili disabilitati
B10	DF	Direction (per operazioni su stringhe)	1 direzione verso il fondo della memoria (decremento DI/SI) 0 direzione verso la sommità della memoria (incremento DI/SI)
B11	OF	Overflow	1 quando il risultato supera il limite del registro 0 altrimenti

Tab. 2 Il registro di stato dell'8086 e il significato dei bit («F» sta per «flag»).

## 4 L'organizzazione della memoria

La CPU 8086 dispone di un bus di indirizzi a 20 linee, in grado di indirizzare fisicamente  $2^{20}$  byte, cioè 1 megabyte (MB) di memoria, che in notazione esadecimale vanno dall'indirizzo 00000h all'indirizzo FFFFh. Le prime 16 linee del bus di indirizzi (0...15) servono anche per indirizzare le periferiche coinvolte nelle operazioni di Input/Output, che al massimo sono  $2^{16}$ , cioè 65 536.

### ■ La segmentazione della memoria

Gli indirizzi delle celle di memoria sono di 20 bit, mentre i registri sono di 16 bit. Come possono i registri puntatori, come per esempio l'Instruction Pointer, contenere un indirizzo?

Il segreto è nel modo in cui l'8086 gestisce la memoria, che è divisa in segmenti. Un segmento è una porzione di RAM utilizzato o per i dati o per le istruzioni. La dimensione di ogni segmento è di 64 kilobyte (kB) al massimo, cioè  $2^{16}$  locazioni (dall'indirizzo 0000h all'indirizzo FFFFh) che possono essere indirizzate con 16 bit.

Per indirizzare una locazione di memoria si indica il segmento che la contiene e all'interno del segmento la locazione desiderata. Questo può essere fatto utilizzando un indirizzo che prevede:

- una *base*, che è l'indirizzo di inizio del segmento;
- uno *spiazzamento*, che è l'indirizzo della locazione a partire dall'inizio del segmento.

### ■ Pit Stop

- 4 Che cosa si intende per «segmento»?
- 5 Da quali parti è composto un indirizzo logico?

Al livello del *software* viene definito un indirizzamento segmentato (logico) costituito da due numeri di 4 cifre esadecimali ciascuno:

- il primo numero costituisce la base;
- il secondo numero costituisce lo spiazzamento (offset).

L'indirizzo logico è espresso con una notazione della forma *base:spiazzamento*.

### Esempio 1

- 3AB0 è la base, cioè l'indirizzo di inizio di un segmento;
- 67CE è lo spiazzamento, cioè la distanza dalla base.



A partire da un indirizzo logico è possibile calcolare l'indirizzo fisico associato:

- si moltiplica la base per 10h (che corrisponde a 16 decimale), cioè si aggiunge uno 0 a destra;
- si somma lo spiazzamento.

Questa trasformazione è a carico della CPU, che preleva dal programma l'indirizzo logico, lo trasforma in indirizzo fisico e lo pone sul bus degli indirizzi, come è mostrato nella Figura 5.

### ■ notabene

Per moltiplicare per 16 (cioè per la base) un numero esadecimale, è sufficiente spostarne a sinistra di un posto tutte le cifre e aggiungere uno zero nel primo posto a destra.

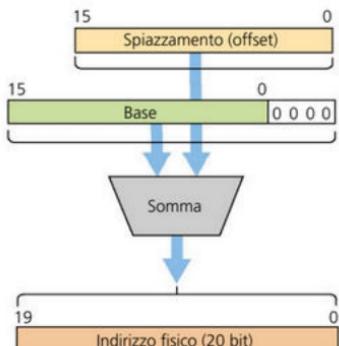


Fig. 5 Calcolo dell'indirizzo fisico a partire da un indirizzo logico: la base è moltiplicata per 16 (10h) e viene sommato l'offset. Moltiplicare per 16 equivale ad aggiungere quattro bit a zero.

### Esempio 2

3AB0:0003		indirizzo logico
3AB0h * 10h = 3AB00h+		
0003h=		
<u>3AB03h</u>		indirizzo fisico

Si noti che 3AB03 è un numero esadecimale di 5 cifre, che corrispondono a 20 cifre binarie (bit).

### Esempio 3

1000:0011		indirizzo logico
10000h+		
0011h=		
<u>10011h</u>		indirizzo fisico

Indirizzi logici diversi possono dar luogo a un unico indirizzo fisico (come nell'esempio precedente).

### Comprendi con l'analogia

L'indirizzo relativo è come la posizione di un quadro appeso in un appartamento calcolata rispetto al piano dell'appartamento, invece che rispetto al suolo

Al secondo piano del palazzo mostrato in figura è appeso un quadro che, relativamente al pavimento del piano, è posizionato a 1 m di altezza (offset o spiazzamento). Rispetto al suolo il quadro si trova invece a 7 m (posizione fisica).



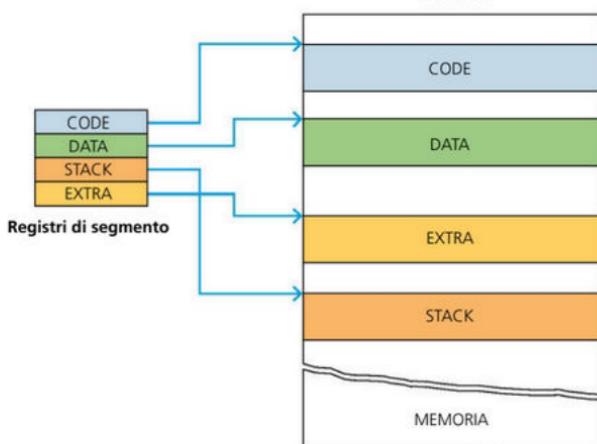
### ■ Tipi di segmento

I segmenti in cui è divisa la memoria sono di quattro tipi ([Figura 6](#)).

Come abbiamo visto, per indirizzare una locazione all'interno di un segmento, il microprocessore 8086 utilizza un indirizzamento logico costituito dalla base più uno spiazzamento.

Per esempio, l'indirizzo logico B200:23F1 significa: locazione di indirizzo 23F1 a partire dalla locazione di indirizzo B200. La base è costituita da un registro di segmento (CS, DS, ...), mentre lo spiazzamento indica la distanza dall'inizio del segmento.

**Figura 6** I segmenti di memoria dell'architettura 8086. Ogni segmento inizia all'indirizzo contenuto nel relativo registro dei segmenti.



Di seguito diamo la descrizione dei segmenti e l'indicazione dei registri utilizzati per lo spiazzamento ([Tabella 3](#)).

↓ Tab. 3

Nome segmento	Registro che contiene l'indirizzo di inizio	Contenuto del segmento	Registri utilizzati per lo spiazzamento
<b>Code Segment</b>	<b>CS</b> È inizializzato dal Sistema Operativo.	<b>Istruzioni</b> È l'unico segmento che contiene il codice.	Lo spiazzamento è posto nel registro IP ( <i>Instruction Pointer</i> ). L'indirizzo sarà quindi della forma CS:IP. Al reset CS è inizializzato con FFFFh e IP con 0000h (indirizzo fisico FFFF0h).
<b>Data Segment</b>	<b>DS</b> Deve essere inizializzato dal programmatore all'interno del suo programma.	<b>Dati</b> Contiene i dati utilizzati dal programma.	Il registro SP (puntatore alla cima dello stack) e il registro BP (puntatore alla base dello stack). L'indirizzo sarà quindi della forma SS:SP, SS:BP.
<b>Stack Segment</b>	<b>SS</b> È inizializzato dal Sistema Operativo.	<b>Dati</b> Contiene i dati gestiti nello stack.	I registri SI, DI, BX. L'indirizzo sarà quindi della forma DS:SI, DS:DI, DS:BX.
<b>Extra Segment</b>	<b>ES</b> Deve essere inizializzato dal programmatore all'interno del suo programma.	<b>Dati</b> Contiene dati particolari usati dal programma.	I registri SI, DI. L'indirizzo sarà quindi della forma ES:SI, ES:DI.

L'inizializzazione dei registri di segmento che contengono dati (DS, ES) è a carico del programmatore e non è ammesso l'indirizzamento immediato; non è possibile scrivere `mov DS, valore` (indirizzamento immediato), ma occorre passare dal registro AX (vedi l'[Esempio 4](#)).

### Esempio 4

#### Inizializzazione dei registri DS e ES

```
mov AX,300h
mov DS,AX      ; carica in DS il valore 300h
mov ES,AX      ; carica in ES il valore 300h
```

### Esempio 5

#### Utilizzo del segmento dati

```
mov AX,1000h
mov DS,AX
mov SI,0100h
mov DI,0200h
mov BX,0300h
indirizzo logico    indirizzo fisico
DS:SI → 1000:0100 → 10100h
DS:DI → 1000:0200 → 10200h
DS:BX → 1000:0300 → 10300h
```

- Con questi registri DS è implicito (di default).
- A volte DI è usato con ES al posto di DS.

## Rilocazione dinamica dei programmi

La segmentazione della memoria permette che la scrittura dei programmi sia indipendente dalla loro posizione in memoria. Si parla quindi di **programma rilocabile dinamicamente**.

Ci consente ai sistemi in multiprogrammazione un utilizzo effettivo della memoria disponibile, in quanto i programmi inattivi possono risiedere su disco, liberando spazio nella memoria centrale. Essi verranno poi caricati in un qualsiasi spazio disponibile all'occorrenza. Non c'è quindi una dipendenza dall'indirizzo di memoria in cui è allocato un programma.

Consideriamo, per esempio, un programma costituito da N istruzioni:

```
istruzione 1
istruzione 2
istruzione 3
...
istruzione N
```

#### notabene

I sistemi multiprogrammati possono eseguire più processi «in parallelo».

Sappiamo che le istruzioni devono essere eseguite in sequenza e quindi allocate in locazioni consecutive. La rilocabilità consente di caricare l'intero programma a partire da indirizzi di memoria liberi. In questo modo basta modificare la base dell'indirizzo relativo, ma lo spiazzamento (che è la parte usata nelle istruzioni software, come vedremo tra poco) non cambierà: per esempio, la terza istruzione

sarà sempre la terza istruzione, relativamente all'inizio del programma. È come se numerassimo le pagine di un libro con una numerazione del tipo *numero capitolo/numero pagina*: la 54-esima pagina di un capitolo sarà sempre la 54-esima pagina all'interno del capitolo, anche se si spostasse il capitolo intero all'interno del libro.

Questo modo di utilizzare la memoria permette di ottimizzarne l'impiego.

## 5 I pin della CPU 8086

La Figura 7 mostra i collegamenti esterni della CPU 8086.

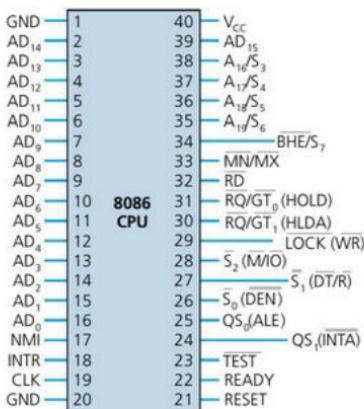


Fig. 7 Collegamenti esterni della CPU 8086: il pinout.

Si possono distinguere 40 segnali, facenti capo ai relativi pin numerati:

- le 20 linee del **bus indirizzi** (A<sub>0</sub>, ..., A<sub>19</sub>);
- le 16 linee del **bus dati** (in comune con il bus indirizzi) (AD<sub>0</sub>, ..., AD<sub>15</sub>);
- le linee del **bus di controllo**, tra cui:
  - M/IO (Memory/Input-Output);
  - RD (Read) WR (Write);
  - INTA (INTerrupt Ack: per il riconoscimento degli interrupt mascherabili);
  - INTR (INTerrupt Request, per la richiesta di interrupt mascherabile);
  - NMI (Not Mascherable Interrupt, per l'invio di interrupt non mascherabile);
  - BHE (per il trasferimento a 8 o 16 bit);
  - CLK (il clock per la temporizzazione delle operazioni).

## 6 Il multiplexaggio temporale di dati/indirizzi

Il bus dei dati condivide i pin con 16 linee del bus indirizzi. I pin sono nominati AD perché sono utilizzati sia per l'indirizzo (A = Address) sia per i dati (D = Data) in tempi diversi. Questa tecnica di **multiplexaggio temporale** di dati/indirizzi (**ADDR/DATA**) permette di interpretare i segnali che fanno capo agli stessi pin come indirizzi o come dati, a seconda del momento in cui vengono presentati sul bus.

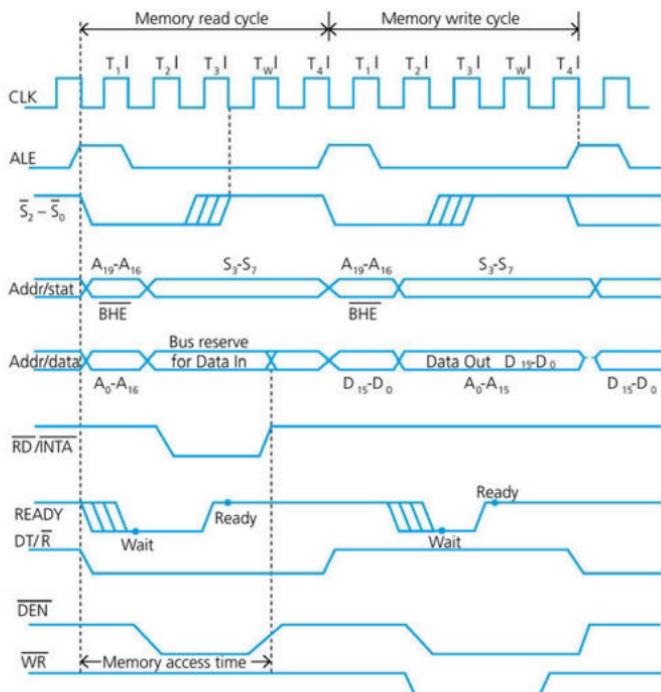


Fig. 8 Multiplexaggio temporale di dati/indirizzi.  
I segnali soprasegnati si intendono «attivi bassi».

Nella Figura 8 è mostrato il diagramma temporale dei segnali per la gestione dei bus:

- **ALE (Address Latch Enable):** specifica esattamente il momento in cui gli indirizzi si alternano ai dati. Nella progettazione dei circuiti con il microprocessore 8086, il segnale ALE viene utilizzato spesso per abilitare il **latch 8282** che cattura gli indirizzi che transitano sul bus. A loro volta i dati che vengono incanalati sul data bus, sono adattati tramite il **ricetrasmettitore 8288**, che è abilitato dal segnale **DEN (Data ENable)**;
- **DT/R (Data Transmit/Receive):** è usato per controllare la direzione di flusso di dati sul bus: in uscita, *Data Out*, e in ingresso, *Data In*;
- **WR (Write):** indica che il processore sta eseguendo una scrittura in memoria o su I/O.
- **RD (Read):** indica che il processore sta eseguendo una lettura da memoria o da dispositivo di I/O.

## 7 Il software dell'architettura 8086: istruzioni e indirizzamenti

### ■ Sintassi

#### ■ notabene

Le istruzioni e i loro codici operativi saranno descritti nei paragrafi successivi.

Un'istruzione Assembly 8086 prevede:

<codice mnemonico> <operandi>

L'istruzione è lunga al massimo **6 byte**, che specificano il **codice operativo**, il **metodo di indirizzamento** e gli eventuali **operandi**.

Il numero di operandi può essere 0, 1 o 2. Quando sono presenti due operandi il primo è il destinatario, il secondo è la sorgente.

Per esempio:

l'istruzione `ret` può avere 0 operandi;  
 l'istruzione `dec` ha 1 operando;  
 l'istruzione `mov` ha 2 operandi.

La **Tavola 4** mostra esempi di istruzioni Assembly 8086.

Codice Assembly	Codice esadecimale	Lunghezza in byte
<code>mov AX,BX</code>	8B C3	2
<code>dec SI</code>	4E	1
<code>mov CX,0100h</code>	B9 00 01	3
<code>mov [0100h],CL</code>	88 0E 00 01	4

→ Tab. 4

Le istruzioni rispettano una **sintassi** rigorosa. Di seguito le principali regole.

- Nelle istruzioni aritmetico-logiche il risultato è posto nel destinatario.
- Se ci sono due operandi, uno solo può essere una locazione di memoria, l'altro è un registro o un dato immediato: non è ammesso il trasferimento da memoria a memoria.
- La notazione con l'uso di parentesi quadre indica una locazione di memoria.  
Per esempio:
  - `add[SI],BX`: somma il contenuto del registro BX al contenuto della locazione di memoria il cui indirizzo è nel registro SI e mette il risultato nella stessa locazione di memoria puntata da SI;
  - `mov AX,[0300h]`: trasferisce il contenuto della locazione di indirizzo DS:0300h nel registro AX;
  - `mov AX,0300h`: trasferisce il valore 0300h nel registro AX.
- Gli operandi possono essere a 8 bit (byte), o a 16 bit (word) e riguardano:
  - dati immediati;
  - registri;
  - locazioni di memoria.
- Se un'istruzione coinvolge due operandi, questi devono avere la stessa dimensione.  
Il programmatore deve sempre determinare con chiarezza la dimensione degli operandi e aver cura che essa sia sempre indicata in modo univoco alla CPU. La dimensione dei registri infatti è evidente, mentre la dimensione dei dati in memoria dipende dal contesto, poiché potrebbe essere di 8 o 16 bit.

- Le istruzioni a due operandi permettono il trasferimento:
  - da registro a registro;
  - da registro a memoria;
  - da memoria a registro;
  - da dato immediato a registro;
  - da dato immediato a memoria.
- I dati in memoria, in genere, sono contenuti nel segmento *DATA*; a volte, ma per istruzioni particolari, nel segmento *EXTRA* o nel segmento *STACK*. Il nome del segmento di default è *DATA*. Nel caso in cui invece si volesse di proposito utilizzare un altro segmento, occorre specificarlo. Per esempio:
  - mov AL,[SI]: trasferisce il contenuto della locazione posta all'indirizzo DS:SI nel registro AL;
  - mov AL,ES:[SI]: trasferisce il contenuto della locazione posta all'indirizzo ES:SI nel registro AL.
- Le istruzioni possono essere scritte in maiuscolo o in minuscolo. Per esempio:
  - mov AX,3 oppure: MOV ax,3.

In questo libro adottiamo la convenzione di scriverle in caratteri minuscoli.

### ■ Metodi di indirizzamento

I principali metodi di indirizzamento della CPU 8086 sono mostrati nella **Tabella 5**.

Metodi di indirizzamento che non coinvolgono la memoria:	Metodi che coinvolgono la memoria:
<ul style="list-style-type: none"> <li>• immediato</li> <li>• a registro</li> </ul>	<ul style="list-style-type: none"> <li>• diretto</li> <li>• indiretto a registro</li> <li>• indiretto a registro base con spostamento</li> <li>• indiretto a registro base e indice</li> <li>• indiretto a registro base, indice e spostamento</li> </ul>

◀ **Tab. 5** Principali metodi di indirizzamento dell'8086.

### Indirizzamento immediato

L'istruzione contiene l'operando rappresentato da un dato numerico. Quest'ultimo si trova in memoria nel segmento del codice.

### Esempio 6

```

mov AL,03h ; carica 03 in AL
add BX,23A4h ; somma al contenuto di BX il numero 23A4
se CS = 2000h:
  2000:0000  mov AL,03h    B0 → codice operativo
  2000:0001          03 → dato
  2000:0002  add BX,23A4h  81 } → codice operativo
              C3 }         A4 } → dato
              23 }

  2000:0003
  2000:0004
  2000:0005

```

---

```

  mov BX,0410h           ; BX → BH=04h BL=10h
  mov AL,05h
  mov AX,0FFFFh

```

**Indirizzamento a registro**

L'operando/gli operandi sono registri della CPU.

**Osservazioni**

- Gli operandi devono avere la stessa dimensione, cioè essere entrambi a 8/16 bit.
- Non è possibile caricare un dato immediato in un registro di segmento.
- Nessuna istruzione può avere come operando IP.

**Esempio 7**

```
mov CL,AL    ; il contenuto di AL è posto in CL
inc DX      ; il contenuto di DX è incrementato di 1
mov AL,BL    ; il contenuto di BL è posto in AL
mov DS,DX    ; il contenuto di DX è posto in DS
mov AL,11h   ; trasferisce il valore 11h nel registro AL
```

**Indirizzamento diretto**

Nell'istruzione è contenuto l'indirizzo di memoria in cui reperire l'operando. In particolare è specificato l'offset relativo a DS perché, di default, i dati sono nel segmento *DATA*. Nel caso fosse necessario utilizzare un altro segmento, occorre indicarne esplicitamente il segmento nell'istruzione.

Per completare l'indirizzo la CPU usa DS (di default), altrimenti occorre indicare nell'istruzione quale registro base utilizzare (OVERRIDE).

Generalmente l'offset è espresso con una *label* associata a un indirizzo, ma può essere utilizzato un valore numerico chiuso tra parentesi quadre, così: [ ].

**Esempio 8**

```
mov DL,[1000h]          ; carica in DL il contenuto della locazione di memoria di
                        ; indirizzo di DS:1000

dati DB 0,0,1            ; alloca 3 byte all'indirizzo associato
                        ; all'etichetta "dati" e li inizializza
.....
mov dati,0              ; azzerà la locazione "dati" ("dati" è l'etichetta
                        ; associata al primo byte)
dati SEGMENT
    numero             DB F5h      ; assegna alla variabile "numero" il valore F5h
dati ENDS
datil SEGMENT
    num1               DB? ; alloca 1 byte, ma non inizializza
datil ENDS
.....
ASSUME CS:codice,        DS:dati, ES:datil
.....
mov CH,numero           ; carica in CH il contenuto della locazione di
                        ; indirizzo DS:numero (carica in CH il valore F5h)
mov ES:num1,CH           ; copia il contenuto di CH nella locazione di
                        ; indirizzo ES:num1 (override)
```

**Indirizzamento indiretto a registro**

Nell'istruzione è contenuto un registro che contiene l'indirizzo di memoria (offset) in cui reperire l'operando.

L'offset si trova in:

- SI relativamente al registro DS;
- DI relativamente al registro DS;
- BX relativamente al registro DS;
- BP relativamente al registro SS.

Il registro contenente l'offset è espresso tra parentesi quadre [ ].

Nel registro deve essere caricato prima lo spiazzamento, utilizzando l'istruzione **LEA (Load Effective Address)** oppure la direttiva offset, come è mostrato nell'esempio. lea e offset restituiscono l'indirizzo in cui è allocato un dato, indirizzo che non è mai noto a priori.

### Esempio 9

```
num DW, 5A00h ; assegna alla variabile "num" il valore 5A00h
.....
lea SI,num      ; carica in SI l'indirizzo di inizio della variabile num
mov AX,[SI]      ; copia in AX la word 5A00 presente in DS:num
```

### Utilizzo del registro BX

; memoria puntata da BX (VARA è il nome di una variabile):

```
mov BX,OFFSET VARA ; carica in BX l'indirizzo (offset) di VARA
; oppure:
lea BX,VARA        ; carica in BX l'indirizzo di VARA (equivalente
; alla precedente)
mov CX,[BX]          ; carica in CX il contenuto puntato da BX
mov AL,[BX]          ; carica in AL il contenuto puntato da BX
mov [SI],BL          ; carica il valore contenuto in BL all'indirizzo
; contenuto in SI
```

### Indiretto a registro con spiazzamento

L'offset è ottenuto come nell'indirizzamento indiretto a registro (si usano gli stessi registri), ma sommando algebricamente un valore costante (spiazzamento, scostamento, *displacement*).

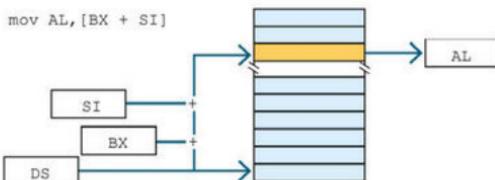
### Esempio 10

Istruzione	Corrisponde a	Descrizione
mov BL,[SI + 5]	BL ← [DS: SI+5]	trasferisce in BL il contenuto della locazione di memoria il cui indirizzo è nel registro SI +5
mov CL,3[SI]	CL ← [DS: SI+3]	trasferisce in CL il contenuto della locazione di memoria il cui indirizzo è nel registro SI +3
mov [SI] + 2, AL	[DS: SI+2] ← BL	trasferisce il contenuto di BL nella locazione di memoria il cui indirizzo è nel registro SI +2
mov CH,num [SI]	CH ← [DS: SI + num]	trasferisce in CH il contenuto della locazione di memoria il cui indirizzo è posto nel registro SI a cui si somma il valore posto all'indirizzo associato a num
mov [SI] num,DH	[DS:SI+num] ← DH	trasferisce il contenuto di DH nella locazione di memoria il cui indirizzo è contenuto in SI a cui si somma il valore di num

**Indiretto a registro base e indice.**

L'offset è ottenuto usando i registri base e indice sommati (Figura 9):

- BX con DS;
- BP con SS.



→ Fig. 9 Esempio di indirizzamento che carica nel registro AL il dato contenuto nel segmento DS e puntato tramite SI e BX.

**Esempio 11**

```

mov BL, [BX+SI]      ; trasferisce in BL il contenuto della
                      ; locazione il cui indirizzo è la somma dei contenuti di BX e SI
mov CH, [SI]+[BX]    ; trasferisce in CH il contenuto della locazione il cui indirizzo è la somma dei contenuti di BX e SI
mov [BX][SI],AL      ; trasferisce nella locazione DS:BX+SI il contenuto di AL
mov [BP][SI],DH      ; trasferisce in SS:BP+SI il contenuto di DH

```

**Indiretto a registro base, indice e spiazzamento**

L'offset è determinato dalla somma dei contenuti dei registri base e indice più un valore aggiunto (spiazzamento).

**Esempio 12**

Istruzione	Corrisponde a
mov BL, [BX+SI+2]	BL $\leftarrow$ DS:BX+SI+2
mov CH, [SI]+[BX] + 5	CH $\leftarrow$ DS:SI+BX+5
mov 4[BX][SI],AL	DS:BX+SI+4 $\leftarrow$ AL
mov num [BP][SI],DH	SS: BP+SI+num $\leftarrow$ DH

La **Tavella 6** mostra il quadro riepilogativo degli indirizzamenti.

Quadro riepilogativo degli indirizzamenti	
Indirizzamento	Registro di segmento
immediato	--
diretto	DS
indiretto tramite BX	DS
indiretto tramite BP	SS
indiretto tramite SI	DS
indiretto tramite DI	DS
indiretto tramite BX + spiazzamento (8 o 16 bit)	DS
indiretto tramite BP + spiazzamento (8 o 16 bit)	SS
indiretto tramite SI + spiazzamento (8 o 16 bit)	DS
indiretto tramite DI + spiazzamento (8 o 16 bit)	DS
indiretto tramite BX + SI	DS
indiretto tramite BX + DI	DS
indiretto tramite BP + SI	SS
indiretto tramite BP + DI	SS
indiretto tramite BX + SI + spiazzamento (8 o 16 bit)	DS
indiretto tramite BX + DI + spiazzamento (8 o 16 bit)	DS
indiretto tramite BP + SI + spiazzamento (8 o 16 bit)	SS
indiretto tramite BP + DI + spiazzamento (8 o 16 bit)	SS
<b>override:</b> ridefinisce il registro base di <i>default</i> .	
<b>esempio:</b> mov AL,ES:[SI]	ES
<b>esempio:</b> mov BL,DS:[BP+300h]	DS

← Tab. 6

## ■ Set di istruzioni

Le istruzioni del software dell'assembly 8086 sono divise in categorie:

- **di trasferimento**
- **aritmetico/logiche**
- **di salto**
- **di rotazione e shift**
- **sulle stringhe**

Di seguito diamo una panoramica di quelle più utilizzate.

### Istruzioni di trasferimento

**Sintassi:**      mov dest,sorg

**Descrizione:** trasferisce (copia) l'operando sorgente in quello di destinazione. Non modifica i flag del registro di stato.

- L'istruzione `mov` permette vari tipi di indirizzamento: immediato, diretto e indiretto.
- Gli operandi di `mov` devono avere entrambi la stessa dimensione (8 bit o 16 bit).
- Spesso gli operandi dell'istruzione `mov` sono i registri di uso generale:
  - AX accumulatore
  - BX base
  - CX contatore
  - DX datia 16 bit

È opportuno ricordare che questi registri possono essere utilizzati a 16 bit o a 8 bit, come è mostrato nella [Tabella 7](#).

16 bit	8 bit parte alta	8 bit parte bassa
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

→ Tab. 7

I registri a 8 bit possono essere utilizzati in modo indipendente, ma si riferiscono al registro completo a 16 bit: il riporto della parte bassa non si propaga nella parte alta.

### Esempio 13

#### Utilizzo dei registri di uso generale a 8 o a 16 bit

```
AX ← A03Fh  AX
   / \
   AH AL
   A0 3F
```

---

```
BH ← 3C      BX ← 3C23
BL ← 23      / \
               BH BL
               3C 23
```

---

```
DX ← 00FF e incrementa il contenuto di DL
DX      DX = 0000
/ \
DH DL
00 FF    FF+1=100h
```

---

```
mov destinazione,sorgente
inc destinazione
AX ← A03Fh :      mov AX, A03Fh
BH ← 3Ch   :      mov BH, 3Ch
BL ← 23h   :      mov BL, 23h
DX ← 00FFh :      mov DX, 00FFh
DL ← DL+1  :      inc DL
```

**Istruzioni aritmetico/logiche**

Istruzioni logiche 8086			
Istruzione	Destinazione	Significato	Esempio
and dest,sorg	dest = dest and sorg	AND logico bit a bit	mov AL, 01100001b ; AL = 01100001b and AL, 11011111b ; AL = 01000001b
or dest,sorg	dest = dest or sorg	OR logico	mov AL, 'A' ; AL = 01000001b or AL, 00100000b ; AL = 01100001b
xor dest,sorg	dest = dest xor sorg	OR esclusivo	mov AL, 01100001b ; AL = 01100001b xor AL, 11011111b ; AL = 10111110b
test dest,sorg	Modifica flag senza memorizzare il risultato	Come AND ma influenza solo i flag	mov AL, 00000101b test AL, 1 ; ZF = 0 mov AL, 00000001b test AL, 00000100b ; ZF = 1
not dest	dest	NOT del contenuto dell'operando	mov AL, 00000101b not AL ; AL = 11111010

Istruzioni aritmetiche 8086			
Istruzione	Destinazione	Significato	Esempio
adc dest,sorg	dest = dest + sorg + CF	Somma con riporto (nella somma comprende anche il flag CF)	adc ax,bx adc al,09h adc bx,0458h
add dest,sorg	dest = dest + sorg	Somma senza riporto	add dx,cx add ax,0400h add label,bl
sbb dest,sorg	dest = dest - sorg - CF	Sottrazione con riporto (nella sottrazione comprende anche il flag CF)	sbb ax,cx
sub dest,sorg	dest = dest - sorg	Sottrazione senza riporto	sub AH,03h
inc dest	dest = dest +1	Incrementa l'operando di una unità	inc cl
dec dest	dest = dest - 1	Decrementa l'operando di una unità	dec cl
cmp op1,op2	op1 - op2	Sottrae operando2 a operando1, ma non modifica la destinazione. Modifica il registro di stato. È usata per fare i confronti	cmp al,03h
mul reg/ memoria	se operando è un byte: AX = AL * operando se operando è una word: (DX AX) = AX * operando	Moltiplicazione	mov AL, 23 mov BL, 24 mul BL;(AX=AL*BL)
div reg/ memoria	se operando è un byte: AL = AX / operando AH = resto se operando è una word: AX = (DX AX) / operando DX = resto	Divisione	mov AX, 203 ; AX=00CBh mov BL, 4 div BL ; AL=50 (32h), AH=3 ; esegue 203 : 4 mette il risultato in AL e il resto in AH

**Istruzioni di salto**

Ricordiamo che il registro di stato è costituito da 16 bit, di cui 6 di stato (CF, PF, AF, ZF, SF, OF) e 3 di controllo (TF, IF, DF).

I bit del registro di stato vengono testati nelle istruzioni di salto. La Tabella seguente riassume le istruzioni di salto previste nell'8086 e il loro significato.

Salto condizionato	Testa se...	Flag interessato
jz je	Operando1 = Operando2	Z=1
jnz jne	Operando1 <> Operando2	Z=0
js jl	Operando1 < Operando2	S=1
jns jge	Operando1 > Operando2	S=0
jc jb	Operando1 < Operando2	C=1
jnc jae	Operando1 >= Operando2	C=0
ja	Operando1 > Operando2	C=0 Z=0
jbe	Operando1 <= Operando2	C=1 Z=1
jo	Overflow	O=1
jno	NO overflow	O=0
jp	Parity	P=1
jnp	NO parity	P=0

**Istruzioni di rotazione e shift**

Le istruzioni di rotazione sono: rol, ror, rcl, rcr.

Le istruzioni di shift sono: shl, shr, sal, sar.

**rol** (Rotate Left, rotazione a sinistra)

**Sintassi:** rol dest, conteggio

**Descrizione:** i bit dell'operando (dest) scorrano di una posizione verso sinistra; il bit più significativo viene memorizzato sia in CF (Flag Carry) sia nel bit meno significativo che è rimasto libero dopo lo scorrimento. Lo scorrimento viene ripetuto un numero di volte pari a conteggio.

**Esempi:** rol BL,1

rol AX,CL

**ror** (Rotate Right, rotazione a destra)

**Sintassi:** ror dest, conteggio

**Descrizione:** i bit dell'operando (dest) scorrano di una posizione verso destra; il bit meno significativo viene memorizzato sia in CF (Flag Carry) sia nel bit più significativo che è rimasto libero dopo lo scorrimento. Lo scorrimento viene ripetuto un numero di volte pari a conteggio.

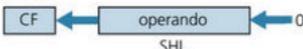
**Esempi:** ror BL,1

ror AX,CL

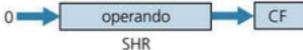
<b>rcl</b>	(Rotate Left through Carry, rotazione a sinistra attraverso il Carry)
<b>Sintassi:</b>	<code>rcl dest, conteggio</code>
<b>Descrizione:</b>	la rotazione verso sinistra dei bit avviene utilizzando il <i>Carry</i> . L'operando che è a 9 o 17 bit (operando più <i>Carry</i> ). Il valore di <i>CF</i> prima della rotazione è significativo.
<b>Esempi:</b>	<code>rcl BL,1</code> <code>rcl AX,CL</code>

<b>rcr</b>	(Rotate Right through Carry, rotazione a destra attraverso il Carry)
<b>Sintassi:</b>	<code>rcr dest, conteggio</code>
<b>Registro di stato:</b>	modifica <i>CF</i> , <i>OF</i>
<b>Descrizione:</b>	la rotazione verso destra dei bit avviene utilizzando il <i>Carry</i> . L'operando che ruota è a 9 o 17 bit (operando più <i>Carry</i> ). Il valore di <i>CF</i> prima della rotazione è significativo. Se <i>conteggio</i> > 1, <i>OF</i> è indefinito. Se <i>conteggio</i> = 1, se dopo la rotazione i due bit più significativi dell'operando sono diversi, <i>OF</i> = 1, altrimenti <i>OF</i> = 0. <i>Conteggio</i> può essere contenuto in <i>CL</i> .
<b>Esempi:</b>	<code>rcr BL,1</code> <code>rcr AX,CL</code>

<b>shl</b>	(Shift Left, scorrimento a sinistra)
<b>Sintassi:</b>	<code>SHL dest, conteggio</code>
<b>Descrizione:</b>	esegue lo scorrimento a sinistra dei bit dell'operando un numero di volte pari a <i>conteggio</i> . L'ultimo bit in uscita viene caricato in <i>CF</i> , mentre le posizioni vuote vengono azzerate. Se <i>conteggio</i> =n, equivale a una divisione per $2^n$ .
<b>Esempi:</b>	<code>shl BL,1</code> <code>shl AX,CL</code>

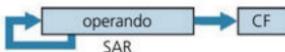


<b>shr</b>	(Shift Right, scorrimento a destra)
<b>Sintassi:</b>	<code>shr dest, conteggio</code>
<b>Descrizione:</b>	esegue lo scorrimento a destra dei bit dell'operando un numero di volte pari a <i>conteggio</i> . L'ultimo bit in uscita viene caricato in <i>CF</i> , mentre le posizioni vuote vengono azzerate. Se <i>conteggio</i> =n, equivale a una moltiplicazione per $2^n$ .
<b>Esempi:</b>	<code>shr BL,1</code> <code>shr AX,CL</code>



<b>sal</b>	(Shift Arithmetic Left, scorrimento aritmetico a sinistra)
<b>Sintassi:</b>	<code>sal dest, conteggio</code>
<b>Descrizione:</b>	esegue lo scorrimento a sinistra dei bit dell'operando un numero di volte pari a <i>conteggio</i> . I bit vuoti a destra vengono riempiti di bit a zero. L'ultimo bit in uscita è copiato in <i>CF</i> . Esegue la moltiplicazione di un numero intero con segno per una potenza di due.
<b>Esempi:</b>	<code>sal BL,1</code> <code>sal AX,CL</code>



**sar** (Shift Arithmetic Right, scorrimento aritmetico a destra)**Sintassi:** sar dest, conteggio**Descrizione:** esegue lo scorrimento a destra dei bit dell'operando un numero di volte pari a conteggio. I bit vuoti a sinistra vengono riempiti di bit pari al valore del bit più significativo. L'ultimo bit in uscita è copiato in CF. Esegue la divisione di un numero intero con segno per una potenza di due.**Esempi:**sar BL,1  
sar AX,CL

### Esempio 14

Il seguente frammento di codice conta il numero di bit a 1 in AL e mette in BL il numero dei bit a 1.

```

mov BL,0
mov CX,8
ciclo: rcr AL,1
        jnc avanti
        inc BL ; trovato bit a 1
avanti: loop ciclo
  
```



**Approfondimento**  
Altre istruzioni 8086

### Istruzioni sulle stringhe

**movs**

```

{REP} movsb
{REP} movsw
{REP} movs Dest,Source
  
```

L'istruzione movsb (sposta una stringa di byte) trasferisce il byte posto all'indirizzo DS:SI (SI ha il significato di sorgente) all'indirizzo ES:DI (DI ha significato di destinazione) e incrementa o decremeta il contenuto dei registri DI e SI (decrementa se DF = 1, incrementa se DF = 0). Se l'istruzione è utilizzata insieme a rep (repeat), viene decrementato anche CX; la CPU testa il registro CX per controllare se è zero. Il ciclo è ripetuto fino a che CX = 0.

### Esempio 15

Il seguente frammento di codice sposta una stringa dal segmento DS al segmento ES. Il numero di caratteri spostati dipende dal valore del registro CX.

```

String1      byte 384 dup (?)
String2      byte 384 dup (?)
            cld
            lea SI, String1
            lea DI, String2
            mov CX, 384
            rep movsb
  
```

**lodsd-lodsb-lods** (Load String, carica una stringa nell'accumulatore)**Sintassi:** lods sorg**Descrizione:** carica nell'accumulatore il contenuto dell'operando sorg posto nel data segment all'offset [SI]; incrementa o decremeta SI in base al valore del Flag DF e alla dimensione del dato (1 o 2 byte).**Esempi:**lods dati  
lodsb

**stos-stosb-stosw** (Store String, carica l'accumulatore in una stringa)

**Sintassi:** stos dest

**Descrizione:** memorizza in dest il contenuto dell'accumulatore (AL o AX, in base alla dimensione) e modifica DI: DI viene incrementato se DF = 0, viene decrementato se DF = 1. È utilizzata con loads. Può essere preceduta da rep per caricare un dato costante in memoria.

**Esempi:**

```
sto ES:dato
rep stosw
```

## ■ Come implementare le strutture di controllo

Vediamo ora come possono essere tradotte le strutture di controllo di un linguaggio di alto livello, per esempio C, in Assembly 8086.

Codice Pseudo-C	Codice Assembly 8086
<b>If...then...else</b>	<pre>mov ax,a cmp ax,b jne Else mov ax,d mov c,ax jmp FineIf Else: inc b FineIf: nop</pre>

Codice Pseudo-C	Codice Assembly 8086
<b>case (switch)</b>	<pre>mov ax,I cmp ax,0 ; confronta il contenuto            ; della variabile I con            ; il valore 0 jne Non0 nop      ; Istruzione _0 jmp FineCase Non0: cmp ax,1 ; confronta il contenuto            ; della variabile I con            ; il valore 1 jne Non1 nop      ; Istruzione _1 jmp FineCase Non1: cmp ax,2 ; confronta il contenuto            ; della variabile I con            ; il valore 2 jne Non2 nop      ; Istruzione _2 jmp FineCase ;... ;... FineCase: ;...</pre>

Codice Pseudo-C	Codice Assembly 8086
<b>while</b> <pre>i = 0; while (i&lt;100) do     i = i + 1; fine while</pre>	<pre>mov I 0 CicloWhile:     cmp I, 10     jge FineWhile     inc I     jmp CicloWhile FineWhile:     nop</pre>

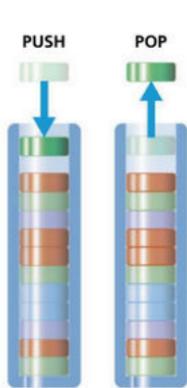


Fig. 10 Gestione dati con politica LIFO.

Codice Pseudo-C	Codice Assembly 8086
<b>for</b> <pre>for (k=1; k &lt; 10; k++) do     totale = totale + 1; fine for</pre>	<pre>mov k,1 CicloFor:     mov al,k     cmp al,10     jge FineFor     inc k     inc totale     jmp CicloFor FineFor:     nop</pre>

### Istruzioni per la gestione dello stack

Lo **stack** o **pila** viene utilizzato per riferirsi a strutture dati le cui modalità d'accesso seguono una politica LIFO (*Last In First Out*), per cui i dati vengono estratti (letti) in ordine rigorosamente inverso rispetto a quello in cui sono stati inseriti (scritti), come è rappresentato nella Figura 10.

La cima dello stack è «puntata» da SP.

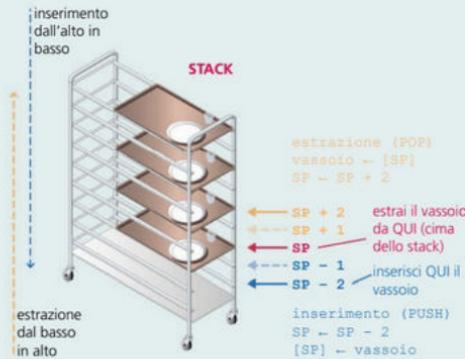
<b>push</b>	carica un operando a 16 bit nello stack
<b>Flag:</b>	non modificati
<b>Sintassi:</b>	push source (Esempi: push AX, push BP...)
<b>Funzionamento:</b>	$SP \leftarrow SP - 2$ $[SP] \leftarrow \text{source}$
<b>Descrizione:</b>	decrementa SP di 2 (perché l'operando è a 16 bit) e copia l'operando nella cima dello stack. L'operando della push non può essere a 8 bit.
<b>pop</b>	preleva un dato a 16 bit dallo stack
<b>Flag:</b>	non modificati
<b>Sintassi:</b>	pop dest (Esempio: pop AX)
<b>Funzionamento:</b>	$\text{source} \leftarrow [SP]$ $SP \leftarrow SP + 2$
<b>Descrizione:</b>	trasferisce un dato a 16 bit (word) dalla cima dello stack (indirizzo SS:SP) al registro destinazione, operando di pop. Poi incrementa SP di 2 affinché punti alla cima dello stack (che «si è abbassata»). Non può essere usato il registro CS come destinazione.

<b>call</b>	richiama procedure
<b>Flag:</b>	non modificati
<b>Sintassi:</b>	call destination (Esempi: call Procedure)
<b>Descrizione:</b>	carica IP nello stack e mette in IP l'indirizzo di inizio della procedura (operando della call). La call riprende poi l'esecuzione dall'indirizzo CS:IP. La procedura chiamata può essere di tipo <b>near</b> , se può essere chiamata solo all'interno del segmento in cui è stata definita (caso di default), o <b>far</b> , se può essere chiamata da qualsiasi segmento.
<b>ret</b>	ritorno da una procedura near
<b>retf</b>	ritorno da una procedura far
<b>Flag:</b>	non modificati
<b>Sintassi:</b>	ret retf
<b>Descrizione:</b>	ritorno da una procedura. Trasferisce il controllo da una procedura all'istruzione il cui indirizzo era stato salvato nello stack. Per le procedure <b>far</b> , la pop preleva IP e subito dopo CS, mentre per le procedure <b>near</b> viene prelevato solo IP (perché CS non è cambiato).

## Comprendi con l'analogia

La gestione dello stack è simile a quella del portavassoi di una mensa

Immaginiamo di riempire il portavassoi di una mensa, seguendo la regola che i ripiani del portavassoi vengano riempiti dal più alto al più basso, senza lasciare spazi vuoti. In tal caso la gestione del riempimento del portavassoi è simile a quella utilizzata per gestire lo stack. Infatti si ottiene lo scopo utilizzando l'algoritmo seguente.



→ Figura 11 La gestione dei vassoi di una mensa utilizzata per riempire il portavassoi.

# Approfondimento - Tecnologia

## Trasferimento delle parole 8086

### L'organizzazione del bus indirizzi, del bus dati e della memoria della CPU 8086

LA CPU 8086 dispone di un bus indirizzi di 20 linee che è quindi in grado di indirizzare fisicamente  $2^{20}$  byte, ossia 1 Mbyte di memoria, dall'indirizzo 00000h all'indirizzo FFFFh. Le prime 16 linee dell'indirizzo (linee dalla 0 alla 15) servono anche per indirizzare le periferiche coinvolte nelle operazioni di input/output. Le periferiche indirizzabili possono quindi essere al massimo  $2^{16}$ , cioè 65536.

Poiché la CPU utilizza un bus dati di 16 linee, è possibile trasferire 2 byte alla volta in un'unica operazione; in questo modo con un unico accesso alla memoria è possibile trasferire 2 byte. Dalla CPU 8086 la memoria è vista come organizzata in due banchi di 512 kilobyte ciascuno. È quindi possibile indirizzare i due banchi insieme (2 byte alla volta), oppure un singolo banco separato (1 byte alla volta).

L'organizzazione della memoria dell'8086 è di tipo little endian, nella quale i byte meno significativi sono memorizzati negli indirizzi più bassi ([Esempio 1](#)).

### Esempio 1

Il valore 1020h, costituito da 2 byte espressi in notazione esadecimale, è memorizzato in questo modo (little endian):

Indirizzo	Dato
0000	20
0001	10

### Il modo in cui si effettuano i trasferimenti tra memoria e CPU

I trasferimenti tra memoria e CPU avvengono nel modo seguente. Riferiamoci all'[Esempio 1](#):

20 → è trasferito tramite le linee  $D_0 \div D_7$  (linee da 0 a 7 del bus dati) e posto negli indirizzi pari di memoria.

10 → è trasferito tramite le linee  $D_8 \div D_{15}$  (linee da 8 a 15 del bus dati) e posto negli indirizzi dispari di memoria.

Ogni locazione di memoria di 16 bit è divisa in 2 byte:

- il byte più significativo (alto) è posto agli indirizzi dispari e utilizza le linee  $D_8 \div D_{15}$ .
- il byte meno significativo (basso) è posto agli indirizzi pari e utilizza le linee  $D_0 \div D_7$ , connessi con le rispettive linee del bus dati come mostrato in [Figura 1](#).

Indirizzi dispari	Banco delle locazioni con indirizzi dispari	Banco delle locazioni con indirizzi pari	Indirizzi pari
0001	byte superiore	byte inferiore	0000
0011			0010
.....			.....

I trasferimenti possono coinvolgere anche le istruzioni; queste sono lunghe da 1 a 6 byte e possono iniziare da un indirizzo pari o da un indirizzo dispari.

Durante la fase di esecuzione di un programma, fra CPU e memoria avviene uno scambio di segnali necessario alla selezione e alla gestione delle singole locazioni di memoria. Tale controllo è effettuato tramite la **linea di indirizzo A<sub>0</sub>** e la **linea di controllo BHE** (*Bus High Enable*), si veda la **Tabella 1**.

Per selezionare il byte più significativo è usato un indirizzo dispari ( $A_0 = 1$ ). Per selezionare il byte meno significativo è usato un indirizzo pari ( $A_0 = 0$ ).

BHE	A <sub>0</sub>	Modalità operative
0	0	trasferimento intera parola
0	1	byte alto da/per l'indirizzo dispari della memoria
1	0	byte basso da/per l'indirizzo pari della memoria
1	1	nessuna selezione

1 (disabilitato) si usa solo la parte bassa del bus dati  
 0 (abilitato) disponibile tutto il bus dati

↑ Fig. 1 Organizzazione della memoria in banchi di byte di indirizzo dispari. La prima locazione è all'indirizzo 0000 (pari), la seconda all'indirizzo 0001 (dispari), la terza all'indirizzo 0010 (pari), e così via.

← Tab. 1 Stati del segnale BHE.

Il tipo di accesso e il fatto che il dato/istruzione sia allineato o disallineato influiscono sulla velocità del trasferimento. Infatti, se l'informazione è a due byte e disallineata (inizia a un indirizzo dispari), sono necessari due accessi alla memoria. Nel caso in cui si verifichi un ripetuto accesso a parole non allineate, il tempo di elaborazione può risultare rallentato. Le prestazioni possono essere ottimizzate memorizzando i dati sempre a partire dall'indirizzo pari.

Inoltre, se la prima istruzione da richiamare è associata a un indirizzo dispari la prima fetch è relativa a una sola locazione, mentre se è associata a un indirizzo pari sono coinvolte due locazioni. Di conseguenza, solo a partire da un indirizzo pari si avrà il riempimento della coda (6 byte), mentre nel caso in cui l'indirizzo iniziale sia dispari dovrà essere atteso lo svuotamento di un secondo byte. Questo perché l'algoritmo di gestione della coda opera su due byte alla volta e l'indirizzamento dispari determina la fase di fetch di un solo byte.

# Laboratorio

## Programmazione in Assembly 8086

Gli esercizi che seguono sono stati realizzati utilizzando l'ambiente di sviluppo (assemblatore e debug) fornito da «emu8086». Con l'emulatore è possibile assemblare un programma ed eseguirlo in un ambiente di lavoro che permette di eseguire le istruzioni *step-by-step (single step)*, posizionare break point per bloccare l'esecuzione del programma, visionare i registri, la memoria e le variabili.

Di seguito mostriamo come utilizzare l'emulatore emu8086 scrivendo e mandando in esecuzione un semplice programma. Proporremo poi alcuni problemi di programmazione e la loro soluzione con Assembly 8086.

### ATTIVITÀ 1

#### Utilizzare un ambiente di sviluppo per programmi Assembly 8086

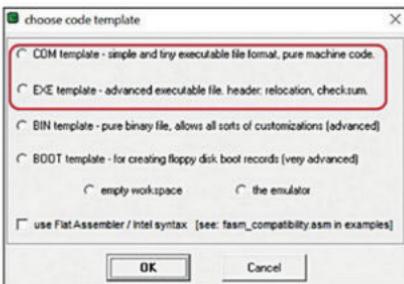
##### Passo 1: installare e aprire emu8086



**Approfondimento**  
Esempi di programmazione in Assembly 8086

##### Passo 2: aprire un nuovo file

8086 supporta vari «modelli di memoria», che possono essere utilizzati per definire l'utilizzo dei segmenti per codice e dati. I formati generalmente utilizzati sono quelli che producono un codice .com e .exe, come è mostrato nella figura. Sebbene i linguaggi ad alto livello (a eccezione del C) mascherino queste scelte, nella programmazione Assembly queste differenze hanno impatto sul codice sorgente che viene scritto.



**Problema guidato**  
Gestione dello stack

Nei file .com si utilizza un unico segmento per dati e codice di 64 kB. I primi 256 byte sono riservati e il programma Assembly deve, necessariamente, partire da 100h.

La direttiva **org 100h** forza inizialmente IP a questo valore. Il puntatore allo stack è inizializzato a FFFEh.

```

new open examples save compile emulate calculator
01 org 100h
02 ; add your code here
03
04 ret
05
06
07
08
09

```

Nei file .exe il programma deve, per prima cosa, impostare i valori di inizio dei segmenti dati, *stack* e codice, utilizzando, rispettivamente, le dichiarazioni «*data*», «*stack*» e «*code*». Il valore di IP è impostato a 0000. Il valore dei registri DS e ES è inizializzato con le istruzioni iniziali di *mov*.

```

file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator converter options
01 ; multi-segment executable file template.
02
03 data segment
04     ; add your data here!
05     pkey db "press any key...$"
06 ends
07
08 stack segment
09     dw 128 dup(0)
10 ends
11
12 code segment
13 start:
14 ; set segment registers:
15     mov ax, data
16     mov ds, ax
17     mov es, ax
18
19     ; add your code here
20
21     lea dx, pkey
22     mov ah, 9
23     int 21h           ; output string at ds:dx
24
25     ; wait for any key...
26     mov ah, 1
27     int 21h
28
29     mov ax, 4c00h    ; exit to operating system.
30     int 21h
31 ends
32
33 end start ; set entry point and stop the assembler.
34

```

Registers	H	L	Value
AX	00	00	07220:
BX	00	00	07221:
CX	01	97	07222:
DX	00	00	07223:
CS	0722		07224:
IP	0000		07225:
SS	0712		07226:
SP	0100		07227:
BP	0000		07228:
SI	0000		07229:
DI	0000		0722E:
DS	0700		0722F:
ES	0700		

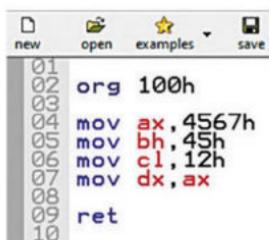
Registers: screen | code

### Passo 3: scrivere il codice

Per semplicità selezioniamo COM.

Sotto alla direttiva **org 100h**, scrivere il codice sorgente e salvare il file sorgente con estensione *.asm*.

Per esempio:



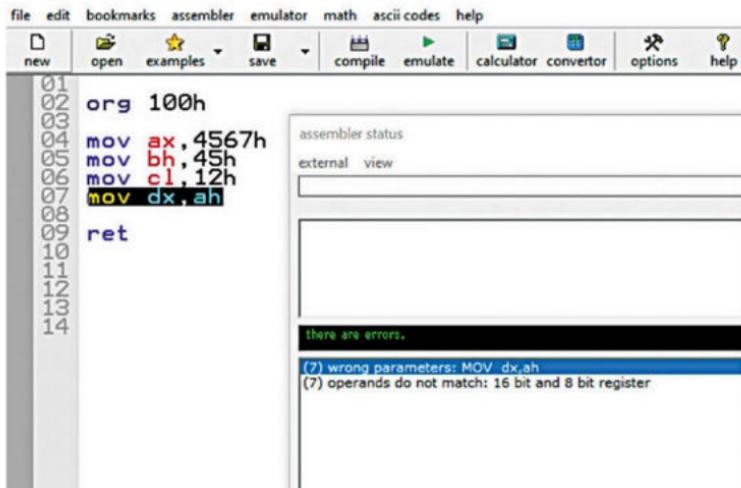
```

01 org 100h
02
03
04 mov ax,4567h
05 mov bh,45h
06 mov cl,12h
07 mov dx,ax
08
09 ret
10

```

#### Passo 4: eseguire l'Assembler

Occorre eseguire l'Assembler per «compilare» il codice sorgente. Selezionare «compile». Eventuali errori saranno segnalati, altrimenti, se non ci sono errori, salvare il file eseguibile. Di seguito è mostrato un esempio di segnalazione di errore dovuto al fatto che gli operandi dell'istruzione alla riga 07 non hanno la stessa dimensione.



The screenshot shows a software interface for assembly language development. The menu bar includes: file, edit, bookmarks, assembler, emulator, math, ascii codes, help. The toolbar includes: new, open, examples, save, compile, emulate, calculator, convertor, options, help. The main window displays assembly code in a text editor. The code is as follows:

```

01 org 100h
02
03
04 mov ax,4567h
05 mov bh,45h
06 mov cl,12h
07 mov dx,ah
08
09 ret
10
11
12
13
14

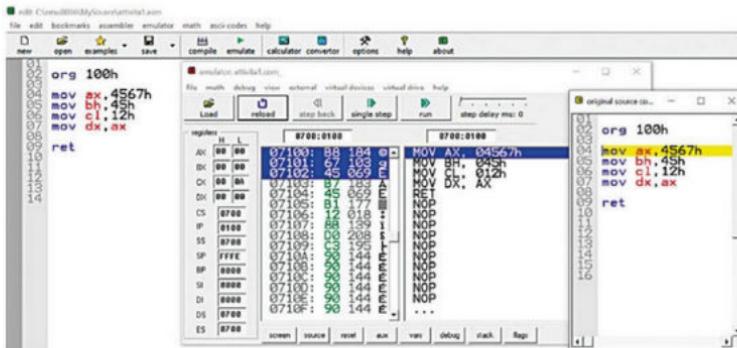
```

To the right of the code editor is a panel titled "assembler status" which contains two tabs: "external" and "view". The "view" tab is active and shows a large empty text area. Below this area is a black bar with the text "there are errors." in white. Underneath is a blue bar containing error messages:

- (7) wrong parameters: MOV dx,ah
- (7) operands do not match: 16 bit and 8 bit register

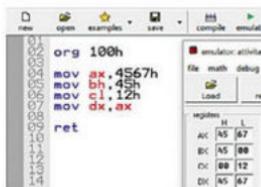
## Passo 5: esecuzione

A questo punto non ci resta che eseguire il programma e verificarne la correttezza. Selezionare *emulate*.



Si aprono due finestre il cui significato sarà illustrato nell'attività 2:

- selezionare *run* per eseguire tutto il codice dalla prima all'ultima istruzione;
- selezionare *single step* per eseguire una istruzione alla volta.



Il programma termina con l'istruzione *ret* che restituisce il controllo al sistema operativo. È possibile verificare la corretta esecuzione del programma osservando il contenuto dei registri.

## ATTIVITÀ 2

Impariamo a leggere il contenuto della memoria e dei registri della CPU

### Obiettivo

Impariamo a leggere il contenuto della memoria e dei registri della CPU.

### Scenario

Riprendiamo il programma dell'attività 1 e osserviamo il contenuto della memoria e dei registri della CPU durante la sua esecuzione. In questo esempio non sono state utilizzate variabili, quindi prenderemo in considerazione solo l'area di memoria relativa al *code segment* (che contiene le istruzioni).

La seguente immagine rappresenta lo stato del sistema dopo l'esecuzione della prima istruzione (`mov ax, 4567h`).

The screenshot shows a debugger window with the following components:

- Registers:** Shows CPU registers AX, BX, CX, DX, IP, CS, SS, SP, BP, SI, DI, DS, ES, and flags (CF, ZF, SF, OF, CS, AF, IF, DF). The CS register is highlighted with a red circle and labeled "Registro di stato con i flag".
- Registers of use:** Shows general and special registers.
- Memory addresses:** Indirizzi di memoria (addresses) from 07100 to 0710F.
- Instructions:** Instruzioni in esadecimale (hexadecimal instructions) starting at address 0700:0100. The first instruction is `MOV AX, 04567h`.
- Code source:** Codice sorgente (source code) on the right side of the window.

### Svolgimento

Osserviamo che:

- il registro **CS** è uguale a **0700**, che significa che il *code segment* inizia all'indirizzo 0700:0000. In questo caso il programma è stato allocato a partire dall'indirizzo logico **0700:0100** che corrisponde all'**indirizzo fisico 07100**;
- la prima istruzione occupa 3 byte e corrisponde al **codice B8 67 45**, dove il primo byte è il codice operativo (B8), gli altri due byte costituiscono l'operando. L'istruzione ha un metodo di indirizzamento immediato;
- la seconda istruzione ha codice B7 45;
- la terza istruzione ha codice B1 12;
- la quarta istruzione ha codice 8B D0;
- il programma occupa 9 byte da 0700:0100 a 0700:0108;
- prima dell'esecuzione della prima istruzione, il registro IP (che ha la funzione di Program Counter) contiene 0100, che è lo spiazzamento rispetto a CS, e costituisce l'indirizzo della prima istruzione da eseguire. Successivamente aumenterà di 3 (lunghezza in byte della prima istruzione) e punterà alla seconda istruzione (IP = 0103), e così via fino al termine del programma.

registers	H	L	
AX	00	00	07100: B8 184
BX	00	00	07101: 67 103
CX	00	0A	07102: 45 069
DX	00	00	07103: B7 183
CS	07	00	07104: 45 069
IP	01	00	07105: B1 177

## ATTIVITÀ 3

### Esempio con variabili e lettura della symbol table e della memoria

#### Obiettivo

Analizzare il contenuto della memoria, nel segmento riservato ai dati, e la symbol table.

#### Scenario

Si consideri il seguente programma che

- somma il contenuto degli elementi corrispondenti di due array di numeri interi e mette il risultato in un terzo array;
- alloca una stringa.

A differenza dell'esempio visto nell'[Attività 2](#), questo programma utilizza delle variabili che saranno allocate nel Data Segment.

```
org 100h

jmp start

pippo db 1,2,3,4
pluto db 3,5,6,1
paperino db ?, ?, ?, ?
clarabella db "ciao a tutti"

start:

lea si, pippo
lea bx, pluto
lea di, paperino

mov cx, 4

sum:
    mov al, [si]
    add al, [bx]
    mov [di], al

    inc si
    inc bx
    inc di

loop sum

ret
```

La direttiva db (*define byte*) alloca spazio di memoria.

In questo caso vengono allocate quattro variabili di nome pippo, pluto, paperino, clarabella.

pippo, pluto, paperino sono tre vettori di quattro numeri interi; clarabella è una stringa che può essere considerata un vettore di caratteri:

- pippo occupa 4 byte ed è inizializzata con i valori 1, 2, 3, 4;

#### notabene

Gli array (vettori) permettono di «accoppare» un certo numero di elementi simili in una singola variabile, rendendo più agevole la manipolazione dei dati.

- pluto occupa 4 byte ed è inizializzata con i valori 3, 5, 6, 1;
- paperino occupa 4 byte, e non è inizializzata perché conterrà il risultato della somma;
- clarabella occupa 12 byte ed è inizializzata con «ciao a tutti».

### Svolgimento

#### Lettura della symbol table

Ricordiamo che la symbol table è una tabella costruita dall'Assembler (quando il sorgente è compilato) che associa il nome delle variabili e delle etichette agli indirizzi di memoria. Usando i nomi sarà possibile utilizzare le variabili indipendentemente dagli indirizzi in cui saranno allocate (per visualizzare la symbol table, utilizzare, nella finestra di debug, le voci **view → symbol table**).

Name	Offset
pippo	00102
pluto	00106
paperino	0010A
clarabella	0010E
start	0011A
sum	00126

#### Lettura della memoria (Data Segment)

Innanzitutto osserviamo che il registro DS contiene 0700, che significa che l'area di memoria riservata ai dati inizia all'indirizzo logico 0700:0000.

Registers	H	L
AX	00	00
BX	01	06
CX	00	32
DX	00	00
CS	0700	
IP	0123	
SS	0700	
SP	FFFE	
BP	0000	
SI	0102	
DI	010A	
DS	0700	
ES	0700	

0700:0123

```

0711A: BE 190 Y - MOV SI, 00102h
0711B: 02 002 E  MOV BX, 00106h
0711C: 01 001 @  MOV DI, 0010Ah
0711D: BB 187 I  MOV CX, 000004h
0711E: 06 006 +  MOV AL, [SI]
0711F: 01 001 @  ADD AL, [BX]
07120: BF 191 N  MOV [DI], AL
07121: 0A 010 NI  INC SI
07122: 01 001 @  INC BX
07123: B9 185 I  INC DI
07124: 04 004 +  LOOP 0126h
07125: 00 000 N  RET
07126: 8A 188 E  NOP
07127: 04 004 +  NOP
07128: 02 002 E  NOP
07129: 07 007 BI  ...

```

screen | source | reset | aux | vars | debug | stack | flags |

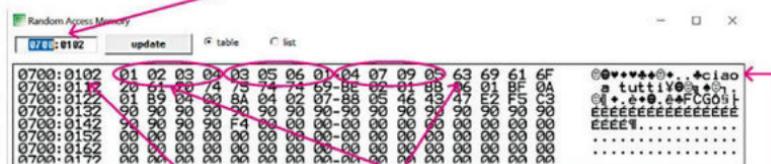
In particolare, ricordando che l'istruzione `lea` (*load effective address*) carica l'indirizzo di una variabile:

- `SI` contiene l'indirizzo di inizio di `pippo` (`lea si, pippo`): `SI=0102`;
- `BX` contiene l'indirizzo di inizio di `pluto` (`lea bx, pluto`): `BX=0106`;
- `DI` contiene l'indirizzo di inizio di `paperino` (`lea di, paperino`): `DI=010A`;
- `clarabella` sarà allocata di seguito.

Apriamo ora la finestra della memoria (**view → memory**) per dare riscontro della symbol table e dei valori delle variabili e dei loro indirizzi.

È possibile posizionarsi in memoria digitando l'indirizzo desiderato.

Nel nostro caso l'indirizzo è: `0700:0102`



Osserviamo che:

- a partire dall'indirizzo `0700:0102` la memoria contiene `01 02 03 04`: variabile `pippo`;
- a partire dall'indirizzo `0700:0106` la memoria contiene `03 05 06 01`: variabile `pluto`;
- a partire dall'indirizzo `0700:010A` la memoria contiene `04 07 09 05`: variabile `paperino` (ogni byte è la somma dei corrispondenti byte di `pippo` e `pluto`: `01+03=04,...`);
- subito dopo `paperino`, all'indirizzo `0700:010E`, è allocata `clarabella`: codice `63`, che è il codice ASCII della «`c`», prima lettera della stringa «`ciao a tutti`».

## ATTIVITÀ 4

### Programmazione Assembly 8086: calcolo di una media

#### Obiettivo

Determinare il valore medio.

#### Scenario

È dato un vettore di 10 numeri interi a 8 bit.

#### Svolgimento

Per semplicità il programma fa la media su variabili a 8 bit; il resto della divisione è salvato nella variabile `resto`.

Casi prova:

- caso 1: `vet={0,1,2,3,4,5,6,7,8,9}`      output: `media=4 resto=5`;
- caso 2: `vet={2,4,6,8,10,12,14,16,18,20}`      output: `media=OB resto=0`.

```

org 100h

jmp start
media    db ?
resto    db ?
;caso1:
vet      db 0,1,2,3,4,5,6,7,8,9
;caso2:
;vet      db 2,4,6,8,10d,12d,14d,16d,18d,20d

start:

mov cx,0    ;azzerà contatore cx
mov cl,10   ; 10 valori

mov al,0
mov media,0 ; azzerà media

lea si,vet
mov bx,0     ; indice del vettore

faisomma:      ; legge i valori e li accumula in media
    mov al,[si][bx]
    mov dl,media
    add al,dl
    mov media,al
    inc bx
    loop faisomma

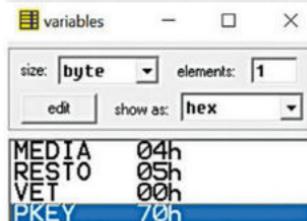
faimedia:
    mov al, media
    mov ah,0
    mov bl, 10 ; divide per 10
    div bl      ;

    mov media,al
    mov resto,ah

ret

```

Al termine dell'esecuzione è possibile osservare il valore delle variabili anche dalle finestre del simulatore:



Provare a variare il programma prelevando i valori da tastiera; ad esempio, sfruttando le istruzioni INT e convertendo il valore ASCII in valore numerico, sottraendo 30H.  
Per esempio è possibile utilizzare i servizi 09 e 01:

```
lea dx, numeri
mov ah,09
int 21h

mov ah,01
int 21H
sub al,30h
```

## ATTIVITÀ 5

### Programmazione Assembly 8086: confronto di variabili

#### Obiettivo

Confrontare il contenuto delle due variabili.

#### Scenario

Sono date due variabili, loc1 e loc2.

#### Svolgimento

Il programma confronta il contenuto di loc1 con quello di loc2. Se sono uguali pone il valore FFh in loc3, altrimenti mette il valore di loc1 in loc3.

Pseudocodice:

```
if (loc1 == loc2) then
    loc3=0FFh
else
    loc3=loc1
fine if
```

Casi prova:

- caso 1: loc1=03, loc2=07      loc3=03;
- caso 2: loc1=07, loc2=07      loc3=0FFh.

```
org 100h
```

```
jmp start
```

```
; dati:
```

```
loc1 db 7
loc2 db 7
loc3 db 0
pkey db "press any key..."
```

```
FLAG _ ON equ 0FFH
```

```
; codice
```

start:

```
    lea bx,loc2      ;trasferisce in bx l'indirizzo di loc2
    mov al,[bx]       ;trasferisce in bl il contenuto di loc2
    mov bl,al
```

```
    cmp bl,loc1      ;confronto il contenuto di loc1 con quel-
    lo di loc2
    je uguali
    diversi:
```

```
        mov al,loc1      ;trasferisce il valore di loc1 in loc3
        mov loc3,al
        jmp exit
```

uguali:

```
        mov loc3,FLAG _ ON ;pone in loc3 il valore FFh
```

exit:ret

## ATTIVITÀ 6

**Programmazione Assembly 8086: conversione di una stringa di caratteri ASCII in maiuscolo e in minuscolo**

### Obiettivo

Convertire la stringa in maiuscolo e in minuscolo.

### Scenario

Abbiamo una stringa di caratteri ASCII.

### Svolgimento

Ricordiamo che nel codice ASCII le lettere minuscole hanno valore numerico minore rispetto alle maiuscole e che per passare da una minuscola alla corrispondente maiuscola occorre sommare 20h.

Per esempio:

- il codice ASCII del carattere «a» è 01100001 (61h);
- il codice ASCII del carattere «A» è 01000001 (41h).

Verificare il funzionamento osservando il contenuto della memoria.

```
org 100h
```

```
jmp start
```

```
string db "123ciao"           ;lunghezza stringa
LENSTR EQU 7
```

```
start:
```

```
;inizializzazione
    lea bx, string
    mov ch, 0
    mov cl, LENSTR
```

```

; controlla se lettera miniscola:
upper _ case:
    cmp [bx], 'a'
    jb go
    cmp [bx], 'z'
    ja go

; converti in maiuscolo
    sub [bx],20h

;oppure e' sufficiente azzerare il bit 5
;esempio:
; 'a' : 01100001b
; 'A' : 01000001b
;and con la maschera      : 11011111b

;    and      [bx], 11011111b

go:
    inc      bx ; prossimo carattere
    loop    upper _ case

;-----
;da lettera maiuscola a minuscola

    lea      bx, string
    mov      ch, 0
    mov      cl, LENSTR

lower _ case:
;controlla se lettera Maiuscola:
    cmp      [bx], 'A'
    jb      gol
    cmp      [bx], 'Z'
    ja      gol

; converti in minuscolo
;e' sufficiente settare il bit 5 a 1
;esempio:
; 'a'           : 01100001b
; 'A'           : 01000001b
;or con la maschera : 00100000b

;    or      [bx], 00100000b

gol:
    inc bx ; prossimo carattere
    loop lower _ case

ret

```

**ATTIVITÀ 7****Programmazione Assembly 8086: il Crivello di Eratostene****Obiettivo**

Codificare in Assembly 8086 un programma che determini i numeri primi  $\leq 20$ .

**Scenario**

Il crivello di Eratostene è un procedimento per determinare i numeri primi da 2 fino a un numero  $n$  prefissato, ideato dal matematico greco Eratostene e ancora utilizzato per il calcolo automatico dei numeri primi.

Ricordiamo che un numero naturale è primo se è divisibile solo per 1 e per se stesso.

Algoritmo:

- si scrivono tutti i numeri naturali da 2 a  $n$  (con  $n$  prefissato);
- si considera il primo numero (2) e si cancellano (setacciano) tutti i multipli di quel numero che non possono essere primi perché divisibili per 2;
- si considera il numero successivo a quello setacciato al passo 1 e si cancellano tutti i suoi multipli;
- si prosegue fino alla fine dei numeri disponibili (la condizione di uscita potrebbe anche essere che il primo multiplo del numero da setacciare è  $> n$ );
- i numeri che rimangono nel setaccio sono i numeri primi  $\leq n$ .

**Svolgimento**

Pseudocodifica:

```
Vet={ 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}
num=20
for(i=2;i<num;i++) {
    for(j=2;j<i;j++){
        if(vett[i+1] % j) == 0 { //numero NON primo
            vett[i+1]=0;
        }
    }
}

; scegliamo il modello exe

LEN equ 20

data segment
    vett db 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
    pkey db "press any key...$"
ends

stack segment
    dw 128 dup(0)
ends
```

```

code segment
start:
; set segment registers:
    mov ax, data
    mov ds, ax
    mov es, ax

    lea      si,vett
    mov     cl,3   ;i

fori:
    mov     dl,2   ;j
forj:
    mov     bh,0
    mov     bl,cl ; bx e' sempre usato come spiazzamento!
    mov     ah,0
    mov     al,[si][bx]      ; (vett+i)
    div
    cmp     ah,0  ; compare resto in dx
    je      nonprimo
salta:
    inc     dl
    cmp     dl,cl
    je      perfori
    jmp     forj

perfori:
    inc     cl
    cmp     cl,LEN
    je      fine
    jmp     fori

nonprimo:
    mov     al,0
    mov     [si][bx],al ; (vett+i)
    jmp     salta

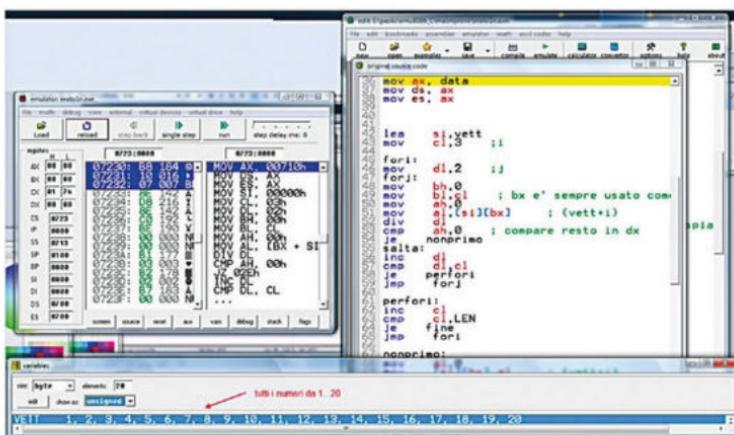
fine:
    lea dx, pkey
    mov ah, 9
    int 21h          ; output string at ds:dx

    ; wait for any key...
    mov ah, 1
    int 21h

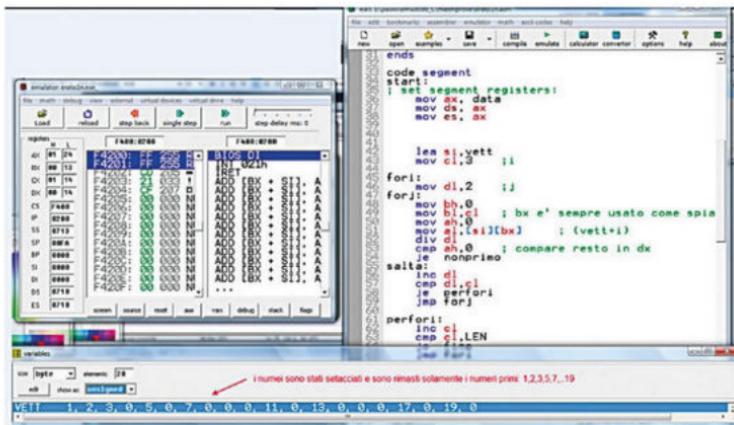
    mov ax, 4c00h ; exit to operating system.
    int 21h
ends

end start ; set entry point and stop the assembler.

```

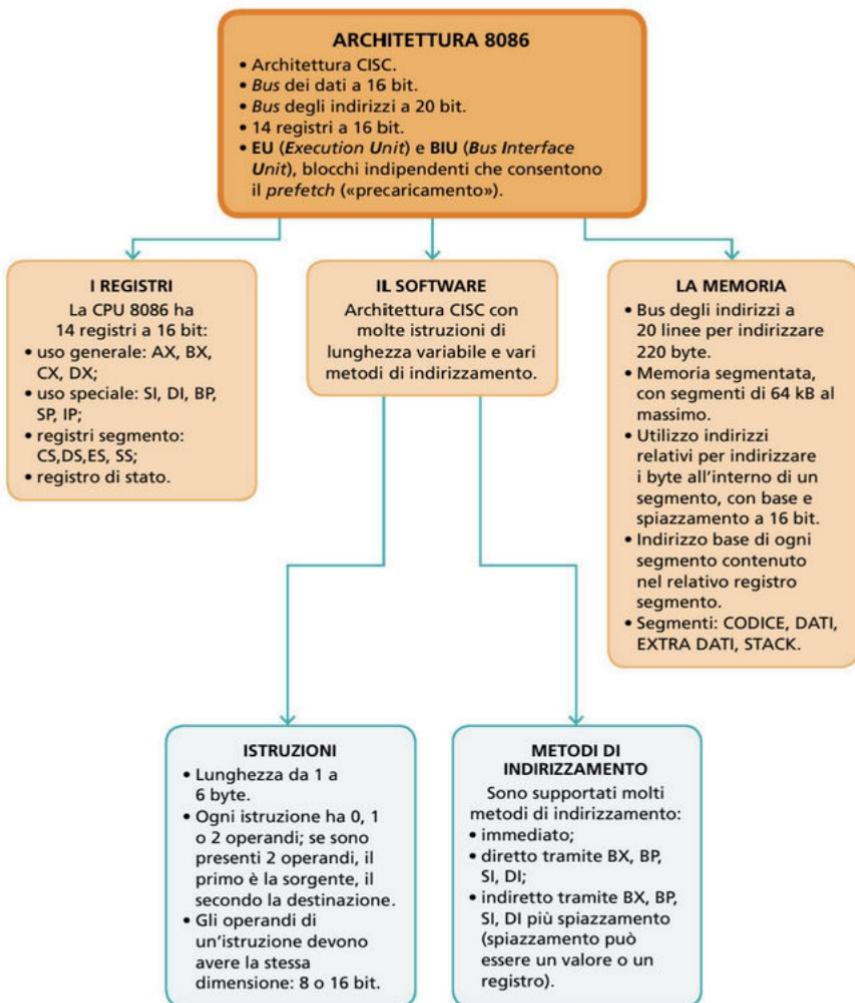


Il programma alla partenza. Nell'immagine sopra è evidenziata, in azzurro, l'area di memoria («VETT») che corrisponde al «setaccio» in cui sono stati caricati i numeri da 1 a 20.



Il programma alla fine dell'esecuzione. Nell'immagine sopra è evidenziata, in azzurro, l'area di memoria («VETT») che corrisponde al «setaccio» in cui sono stati lasciati solo i numeri primi, azzerando tutti gli altri.

# SINTESI





### Vero o Falso?

- 1 Il sistema 8086 è dotato di bus indirizzi a 16 bit.
- 2 L'indirizzo logico identifica in modo univoco una locazione di memoria.
- 3 Il registro IP dell'8086 contiene l'indirizzo fisico della prossima istruzione da eseguire.
- 4 Un segmento dell'8086 è una porzione di RAM.
- 5 Il trasferimento dati in 8086 può essere di 8 o di 16 bit.
- 6 La CPU 8086 ha 36 collegamenti esterni: 16 per il bus dati e 20 per il bus indirizzi.
- 7 Il registro IP è utilizzato con DS per puntare all'istruzione successiva.
- 8 Il registro SS è utilizzato insieme a SP.
- 9 8086 è un'architettura RISC.
- 10 La dimensione massima di un segmento è di 64 kB.



### Rispondi ai seguenti quesiti indicando l'unica risposta esatta.

- 11 L'indirizzo assoluto:
  - a è formato da due numeri di 16 bit
  - b è formato da un numero di 20 bit
  - c serve per indicare un segmento
  - d non identifica in modo univoco una locazione di memoria
- 12 Quali dei seguenti indirizzi 8086 è espresso in forma corretta?
  - a SS:IP
  - b CS:IP
  - c DS:IP
  - d ES:IP
- 13 L'indirizzo logico:
  - a non identifica in modo univoco una locazione di memoria
  - b è associato a più indirizzi fisici
  - c è costituito da un numero di 16 bit
  - d favorisce la rilocabilità di un programma
- 14 La coda di prefetch 8086:
  - a serve per aumentare le prestazioni del sistema
  - b è utilizzata in modo che la EU scrive e la BIU legge
  - c serve solo per le istruzioni di salto
  - d è costituita da 2 byte: uno per il codice operativo e l'altro per l'operando
- 15 Il registro di stato 8086:
  - a ha dimensione 6 bit
  - b serve per segmentare la memoria
  - c ha 9 bit significativi
  - d è usato nelle istruzioni di trasferimento
- 16 Le istruzioni di salto condizionato:
  - a modificano il registro di stato
  - b testano il registro di stato
  - c sono istruzioni di trasferimento
  - d possono essere sostituite dall'istruzione CMP
- 17 L'istruzione mov ax,bh:
  - a ha un indirizzamento diretto
  - b è corretta
  - c non è corretta perché gli operandi non hanno la stessa dimensione
  - d non è corretta perché utilizza un metodo di indirizzamento non previsto
- 18 Il registro CS:
  - a Punta allo stack
  - b Contiene un indirizzo fisico
  - c È la base di un indirizzo logico
  - d È lo spiazzamento di un indirizzo logico

### Domande per la prova orale

- 19 Qual è la differenza tra jp FC00 e call FC00 in 8086?
- 20 Spiega come la CPU esegue un'istruzione call 8086.
- 21 Spiega significato e funzionamento delle istruzioni push e pop 8086.
- 22 Descrivi l'architettura interna della CPU 8086.
- 23 Che cos'è la coda di prefetch? A che cosa serve?
- 24 Descrivi i pin della CPU 8086.
- 25 Descrivi con precisione che cosa fa la CPU per eseguire un'istruzione call.
- 26 Che differenza c'è tra stack e stack pointer?
- 27 Come è possibile per la CPU 8086 «tornare» a eseguire l'istruzione che segue una call?

# Verifica delle ABILITÀ

Unità 7

- 28 Per ogni combinazione di segnali, scrivi una corrispondente istruzione assembly 8086:  
MEM + RD

.....  
MEM + WR  
.....

- 29 Scrivi in linguaggio naturale il significato delle seguenti istruzioni scritte in assembly 8086.

a. mov AX,F324h

b. add BH,03h

c. mov dato,AX

d. inc BL

e. mov DX,[0024h]

f. lodsb

- 30 Scrivi in linguaggio assembly le seguenti istruzioni espresse in linguaggio naturale.

- a. Trasferire nel registro AX il contenuto della locazione di memoria di indirizzo EC34h.  
b. Trasferire nella locazione di memoria di indirizzo 0026h il valore 68h.  
c. Sommare al contenuto del registro AH il valore 64h e mettere il risultato in AH.  
d. Incrementare il valore del registro BL.  
e. Sommare al contenuto della locazione di memoria di indirizzo 0F56h il valore 0002h.  
f. Trasferire nello stack il contenuto del registro AX.

- 31 Scrivi un programma Assembly 8086 per risolvere il seguente problema:  
dato un numero  $N$  da input, scrivere in output i tre numeri successivi.

- 32 Scrivi un programma Assembly 8086 per risolvere il seguente problema:  
dati due numeri  $N$  e  $P$  da input, scrivere in output il risultato di:

$$2 \cdot [(N + 1) + (P - 1)]$$

Provare il programma con i seguenti dati di input:

caso1:  $N = 3, P = 5$

caso2:  $N = 3, P = 7$

caso3:  $N = 2, P = 1$

- 33 Dato il seguente algoritmo, tradurlo in linguaggio Assembly 8086:

Leggi  $N$   
Leggi  $P$   
if ( $N == 7$ ) then  
     $N = P+1$   
fine if  
scrivi  $N$   
scrivi  $P$

Provare il programma con i seguenti dati di input:

caso1:  $N = 3, P = 7$

caso2:  $N = 7, P = 8$

caso3:  $N = 2, P = 3$

- 34 Scrivere un programma 8086 per codificare il seguente algoritmo

leggi  $N$   
leggi  $P$   
if ( $N != P$ ) then  
     $N=N+1$   
else  
     $P=P+1$   
fine if  
scrivi  $N$   
scrivi  $P$

Provare il programma con i seguenti dati di input:

caso 1:  $N = 3, N = 7$

caso 2:  $N = 3, N = 3$

- 35 Considera il seguente programma scritto in linguaggio ad alto livello (in questo caso Java) e riscrivilo in Assembly 8086.

```
public class Assembly {  
    public static void  
    main(String[] args) {  
        int N=0;  
        int i;  
        for (i=0; i<5; i++) {  
            N +=2;  
        }  
        System.out.print( N );  
    }  
}
```

- 36 Scrivi un programma Assembly 8086 per risolvere il seguente problema:

continuare a chiedere all'utente un numero;  
smettere di chiedere quando l'utente inserisce il 7.

- a. Scrivere in output il numero di tentativi fatti.  
b. Scrivere in output il numero di zeri inseriti come tentativo.



## 8086 Architecture

### Abstract

The 8086 CPU is the ancestor of the x86 Intel family. A modern PC functioning is still founded on the 8086 assembly.

**8086 hardware.** The 8086 CPU is a 16-bit microprocessor with 16-bit registers and 16-bit data bus. The 8086 has a data bus multiplexed with an external 20-bit address bus, that can manage 1 MB of physical memory. The memory is managed by «segmentation», with four segment types: data,



extra data, stack and code. Each segment has a max dimension of 64 kbytes so, inside a segment a 16-bit address can be used. The CPU can transform logical addresses, limited to 16 bit, to wider physical addresses.

**8086 software.** The 8086 instructions set is composed by codes with max length of 6 digits, with several addressing methods.

### Questions

#### Answer all questions

- 1 What are the 8086 CPU registers?
- 2 How is the status register of the 8086 CPU made?
- 3 Describe a generic 8086 CPU instruction.

#### Choose the correct answer



- 4 The Zero Flag 8086:
  - a will be set to 1 if the result of the last arithmetic/logic operation is 0
  - b will be set to 1 if the result of the last arithmetic/logic operation is 1
  - c will be set to 1 if the result of the last transfer operation is 0
  - d will be set to 1 if the result of the last transfer operation is 1

#### 5 Choose the right 8086 address:

- a ES:SP
- b DS:IP
- c CS:IP
- d SS:DX

#### 6 What is wrong in 8086?

- a mov ax,bx
- b mov a1,b1
- c mov cl,78h
- d mov ds,78h

#### 7 A physical address in 8086:

- a has 16 bits
- b has 20 bits
- c identifies multiple bytes
- d is made of a base and displacement

### Crosswords

#### Across

4 A special purpose registry used as a program counter

5 A subdivision of memory managed by 8086

7 Part of a memory address

8 Instruction used to start subroutines

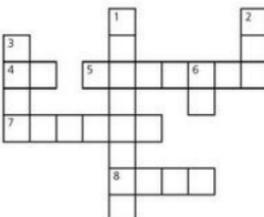
#### Down

1 Load instructions into memory for later use

2 Instruction used to return from a subroutine

3 Management policy of a prefetch queue

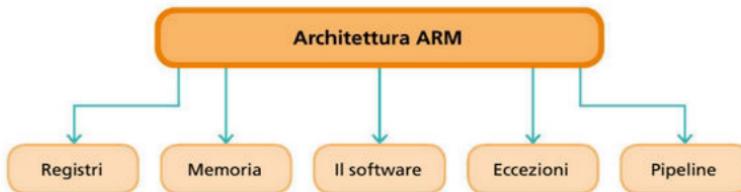
6 Part of a 8086 CPU that executes instructions





# Unità 8

## L'architettura ARM



### Visione d'insieme

- Architettura del microprocessore ARM.
- Registri e gestione della memoria.
- Set di istruzioni, metodi di indirizzamento, gestione delle eccezioni.



Fig. 1 La sede di ARM Holdings nella Silicon Valley.



Fig. 2 MediaTek ARM mobile processor. Il MediaTek MT8312 è un SoC per tablet e smartphone basati su Android. Offre due core del processore ARM Cortex-A7 e una scheda grafica integrata.

## 1 Introduzione

L'architettura ARM comprende una famiglia di microprocessori RISC, sviluppata da **ARM Holdings** (Figura 1), che domina il settore dei dispositivi mobili dove il risparmio energetico è di fondamentale importanza.

Grazie al basso costo, alla velocità di esecuzione e ai consumi ridotti, la famiglia dei microprocessori ARM si è imposta sul mercato mondiale contribuendo al successo dei *System-on-a-Chip*, che sono alla base dei sistemi mobili, dei tablet e delle schede SBC, dei dispositivi di rete e dei sistemi di memoria. ARM Holdings detiene la proprietà intellettuale dell'architettura ARM e del set di istruzioni, ma ci sono molti produttori e molte terze parti che integrano e producono processori ARM (Figura 2).

## 2 L'evoluzione dei processori ARM

Una determinata architettura ARM comprende tipi diversi di processore (core) che elaborano i dati basandosi sullo stesso set di istruzioni.

I primi processori avevano un solo core, ma a partire dal 2010 la necessità di prestazioni elevate ha determinato la progettazione di processori con un maggior numero di core e un crescente grado di parallelizzazione. L'evoluzione ha riguardato diverse famiglie con architettura a 8, 16, 32, 64 bit, con modelli diversi di processori.

La Tabella 1 offre un confronto tra le principali architetture, che rimangono comunque in continua evoluzione.

Famiglia	Architettura	Processore	Caratteristiche
<b>ARM1</b>	ARMv1	ARM1	Capostipite. Pipeline semplice. 26 bit di indirizzamento della memoria. 25 registri a 32 bit
<b>ARM6</b>	ARMv3	ARM60	Il primo a supportare lo spazio degli indirizzi di memoria a 32 bit e aggiungere istruzioni matematiche complesse e coprocessore
<b>ARM11</b>	ARMv6	ARM1136J(F)-S	Pipeline a 8 stadi
<b>Cortex-M</b>	ARMv8.x	Cortex-M33	DSP e coprocessore che aggiungono funzionalità quali l'aritmetica in virgola mobile, la grafica e anche la crittografia. Ogni qualvolta la CPU incontra un'istruzione dedicata a un coprocessore, può chiedergli di eseguirla
<b>Cortex-A (32-bit)</b>	ARMv8-A	Cortex-A32	Cache memory con prestazioni elevate
<b>Cortex-A (64-bit)</b>	ARMv8.2-A	Cortex-A77	Pipeline a 13 stadi, superscalare

Tab. 1

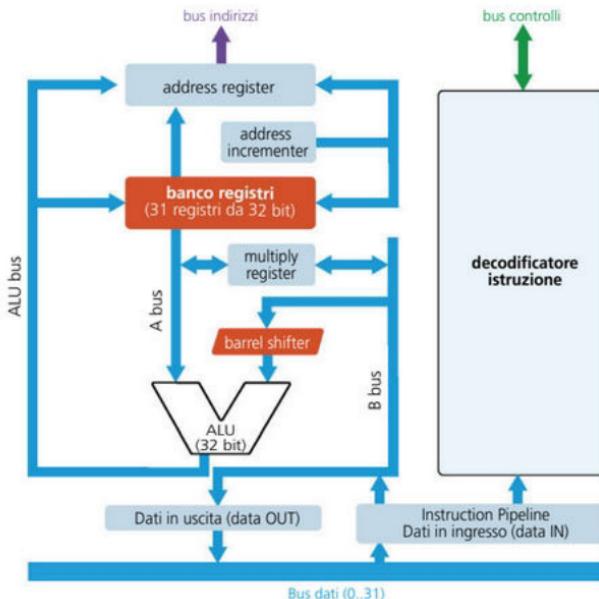
Le CPU ARM comprendono una pipeline a più stadi, con memorie cache separate per istruzioni e dati, e permettono di rendere parallele le fasi di:

1. **Fetch**, per la lettura dell'istruzione;
2. **Decode**, per l'interpretazione del codice operativo e la lettura degli operandi dai registri;
3. **Execute**, per l'esecuzione dell'istruzione;
4. **Buffer/data**, per accedere alla memoria tramite le istruzioni di load e store;
5. **Write-back**, per la scrittura del risultato nel registro di destinazione.

### 3 L'architettura della CPU ARM a 32bit

Iniziamo lo studio di ARM considerando l'**architettura base a 32 bit** illustrata nella **Figura 3**. L'architettura ha le seguenti caratteristiche:

- **registri, istruzioni, bus dati e bus indirizzi** sono tutti a 32 bit. È possibile indirizzare fino a  $2^{32}$  celle di memoria;
- l'**unità barrel shifter**, separata dalla ALU, permette uno *shift* o una rotazione dei bit verso sinistra o verso destra nello stesso ciclo macchina;
- il registro **address incrementer** facilita l'indirizzamento sequenziale alle locazioni di memoria;
- esiste la possibilità di **autoincremento** e **autodecremento** dell'indirizzo, per semplificare i cicli;
- l'architettura **load-store** impone che gli operandi delle istruzioni siano registri e che tutti i dati debbano passare da questi prima di poter essere elaborati;
- le istruzioni in genere utilizzano 3 registri: 2 registri sorgente e 1 registro destinazione;
- è previsto un set standard di istruzioni a 32 bit (e un set di istruzioni a 16 bit);
- è possibile ridurre il numero di istruzioni di salto tramite l'esecuzione condizionale che, riducendo gli svuotamenti della pipeline, ne amplifica l'efficienza.



#### notabene

Le operazioni di shift permettono di far slittare i bit di una posizione a sinistra o a destra.

◀ Fig. 3 La logica di un processore ARM. Registri e bus sono di lunghezza uniforme e fissa a 32 bit. La pipeline fa in modo che vengano parallelizzate le fasi di esecuzione di più istruzioni.

#### Pit Stop

- 1 Quali sono le caratteristiche dell'architettura ARM?

## 4 I registri

Nella modalità di lavoro standard (*User Mode*) sono utilizzabili **16 registri a 32 bit** (**r0, r1, r2, ..., r15**) più il registro di stato chiamato **CPSR** (*Current Program Status Register*, **Figura 4**):

16 registri a 32 bit																
32 bit																
<b>CPSR</b> Status Register	<b>r15</b> Program counter	<b>r14</b> Link Register	<b>r13</b> Stack Pointer	<b>r12</b>	<b>r11</b>	<b>r10</b>	<b>r9</b>	<b>r8</b>	<b>r7</b>	<b>r6</b>	<b>r5</b>	<b>r4</b>	<b>r3</b>	<b>r2</b>	<b>r1</b>	<b>r0</b>

Fig. 4 I registri di ARM.  
**r0, ..., r12** sono registri di uso generale;  
**r13** è usato come Stack Pointer; **r14** è il Link Register; **r15** è il Program Counter; **CPSR** è il registro di stato con i flag.

- **r0, ..., r12** sono registri di uso generale;
- **r13** è utilizzato convenzionalmente come Stack Pointer (anche se l'architettura non lo impone);
- **r14 (LR, Link Register)** è utilizzato per salvare l'indirizzo di ritorno dopo l'esecuzione dell'istruzione **BL (Branch and Link)** che effettua la chiamata a un sottoprogramma; l'indirizzo all'istruzione successiva alla chiamata (**BL + 4**) è posto in **r14**. Per effettuare il ritorno al chiamante occorre copiare **r14** in **r15** (**mov r15, r14** o meglio **mov PC, LR**); se **r14** viene usato durante l'esecuzione del programma come registro di uso generale e poi come **LR**, occorre salvare **r14** prima dell'esecuzione di **BL**;
- **r15 (PC, Program Counter)**.

### Pit Stop

2 Quali sono i registri di ARM?

Tutti questi registri sono accessibili al programmatore e potrebbero essere utilizzati per uso generale, anche se è consigliato l'accesso a **r15**. L'assemblatore associa il simbolo **LR** al registro **r14** e il simbolo **PC** al registro **r15**. Nel caso in cui si utilizzi un assemblatore che non associa questi simboli, conviene definirli con una direttiva all'inizio del programma.

- **CPSR (Current Program Status Register)** è il registro di stato (**Figura 5**).

Fig. 5 I flag del registro di stato **CPSR**.

31	30	29	28	27 ... 8	7	6	5	4	3	2	1	0			
<b>N</b>	<b>Z</b>	<b>C</b>	<b>V</b>		<b>I</b>	<b>F</b>	<b>T</b>						<b>MODE</b>		

**N:** Negative Flag

**I:** distingue tra due modalità di funzionamento (ARM Standard Mode e Thumb Mode):

**Z:** Zero Flag

• **T = 1** Thumb Mode (istruzioni a 16 bit)

**C:** Carry Flag

• **T = 0** ARM Standard Mode (istruzioni a 32 bit)

**V:** overflow Flag

**I, F:** abilitano le interruzioni normali (I) e veloci (F).

**Mode 4 ... Mode 0:** abilitano una modalità di funzionamento

### notabene

L'utilizzo delle lettere minuscole o maiuscole per i registri non ha importanza nel linguaggio Assembly. Scrivere **MOV R0, #1** o **mov r0, #1** è quindi la stessa cosa. In questo libro adottiamo la convenzione di utilizzare sia per le istruzioni che per i registri le lettere minuscole.

## 5 Modalità di funzionamento del processore

Per proteggere parti di memoria o risorse particolari, un processore può operare in diverse modalità utilizzando meccanismi di protezione che offrono dei criteri per controllare le risorse del sistema.

ARM è molto flessibile in questo senso, perché permette di utilizzare fino a **7 modi di esecuzione**.

Seppure la maggior parte delle applicazioni utilizzi la **modalità utente (User Mode)**, che non dispone di privilegi particolari, vi sono casi, come la gestione delle interruzioni hardware e software, che eseguono operazioni speciali (**Tabella 2**) e accedono a registri aggiuntivi.

Infatti, mentre tutte le modalità possono utilizzare i registri r0, ..., r15 e il registro CPSR, solo le modalità privilegiate possono gestire anche registri aggiuntivi, quelli che sono indicati in colore nella **Figura 6**.

	<b>Processor Mode</b>	<b>Mode Number</b>	<b>Descrizione</b>
Modo non privilegiato	User (utente)	0b10000	Modalità standard utilizzata per la normale esecuzione delle applicazioni che vengono eseguite in modo isolato e protetto
Modi privilegiati	FIQ (Fast Interrupt Request)	0b10001	Si entra in questa modalità quando il processore riceve un segnale di interrupt ad alta priorità ( <i>FAST</i> ) che non può essere interrotto
	IRQ (Interrupt Request)	0b10010	Si entra in questa modalità quando il processore riceve un segnale di interrupt a bassa priorità ( <i>NORMAL</i> )
	Supervisor (Supervisore)	0b10011	Si entra in questa modalità al <i>reset</i> o quando il processore incontra un'istruzione di <i>interrupt software (SWI)</i> , che è il tipico modo di invocare una funzione di sistema operativo
	Abort	0b10111	È utilizzato per gestire violazioni di accesso alla memoria
	Undefined (Non definito)	0b11011	Si entra in questa modalità quando il processore tenta di eseguire un'istruzione non definita che non è supportata dall'architettura dei processori
	System (Sistema)	0b11111	Modalità privilegiata che utilizza gli stessi registri di <i>User Mode</i> . È utilizzata per eseguire task privilegiati del sistema operativo

↑ **Tab. 2** Modalità di funzionamento del processore ARM. Ricordiamo che l'architettura ARM è utilizzata per sistemi embedded che molto spesso devono reagire alle interruzioni dovute a eventi esterni ad alta criticità.

<b>Sistema e Utente</b>	<b>Fast Interrupt</b>	<b>Supervisore</b>	<b>Abort</b>	<b>Interrupt</b>	<b>Non definito</b>
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13 (SP)	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
r14 (LR)	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)
<b>Program Status Register</b>					
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

↑ **Fig. 6** Registri disponibili nelle varie modalità di funzionamento del processore. In colore sono evidenziati i 20 registri duplicati, utilizzati nelle diverse modalità di funzionamento. In particolare si noti il registro SPSR (*Saved Program Status Register*), utilizzato per salvare CPSR durante un interrupt software.

**notabene**

Si noti che quando i valori sono espressi in binario o in esadecimale, viene utilizzata la notazione seguente: % o

**0b:** indica che il valore è espresso in binario. Per esempio: %11110000  
o 0b11110000.  
**0x:** indica che il valore è espresso in esadecimale. Per esempio: 0xCAFE.

## 6 La memoria

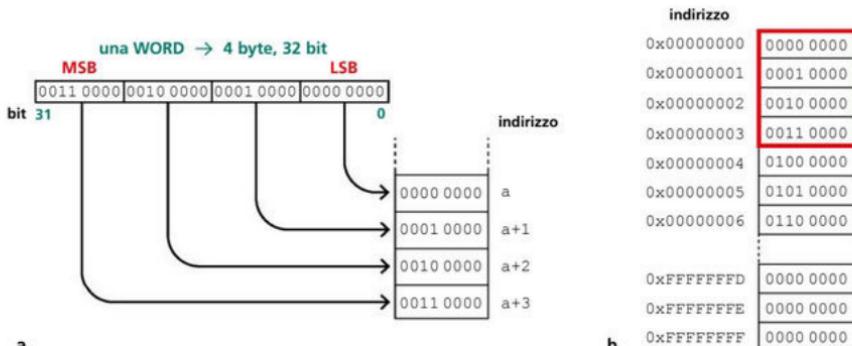
### Organizzazione della memoria

I dati in memoria sono disposti secondo la convenzione **little endian** organizzati in parole (word) da 32 bit (4 byte). La [Figura 7](#) mostra l'organizzazione dei blocchi di memoria di 4 byte.

- Il primo blocco è costituito dai byte 0000 0000, 0001 0000, 0010 0000, 0011 0000;
- il secondo blocco dai byte 0100 0000, 0101 0000, 0110 0000, 0111 0000 e così via ([Figura 7a](#)).

Un indirizzo che corrisponde all'inizio di una parola è chiamato **word boundary** ed è **word aligned** («allineato alla parola»). Da questo deduciamo che tutti gli indirizzi word aligned sono divisibili per 4; per esempio: 0x000009030 0x000009034 0x000009038 0x00000903C.

Questi indirizzi sono particolarmente importanti perché **tutte le istruzioni ARM devono essere caricate in locazioni di indirizzi allineati** ([Fig. 7b](#)).



a.

b.

Fig. 7 Organizzazione della memoria little endian: il byte meno significativo viene posto all'indirizzo più basso. a. Le parole (word) sono composte da 4 byte (32 bit). Il byte meno significativo (LSB) è posto all'indirizzo a, mentre il più significativo (MSB) all'indirizzo a + 3. b. I blocchi sono allineati e non è possibile modificarne l'organizzazione, anche se è possibile accedere al singolo byte.

### Utilizzo della memoria

La [Tabella 3](#) mostra come viene utilizzata la memoria.

indirizzi	utilizzo
0-FFF <sub>hex</sub>	Sistema operativo
1000 <sub>hex</sub> - x	Programma (codice e dati)
x + 1 - 5400 <sub>hex</sub>	Stack
5401 <sub>hex</sub> - 11400 <sub>hex</sub>	Heap (per le strutture dati dinamiche)

→ Tab. 3 Utilizzo della memoria.

- Il registro r13 (SP) viene inizializzato con l'indirizzo della cima dello stack (ultima locazione libera) 5400<sub>hex</sub>.
- Il registro r15 (PC) viene inizializzato con l'indirizzo della prima istruzione 1000<sub>hex</sub>.

### Pit Stop

- 3** Com'è organizzata la memoria di ARM?
- 4** Perché le istruzioni ARM devono essere caricate a indirizzi allineati?

## 7 Il software ARM

Il **set di istruzioni** del processore ARM prevede istruzioni a 32 bit che manipolano dati a 32 bit.

L'utilizzo delle lettere maiuscole o minuscole per istruzioni e registri non è rilevante.

Tutte le istruzioni operano sui registri, a eccezione delle istruzioni che operano sulla memoria.

Le istruzioni si dividono in:

- istruzioni **data processing** o di elaborazione dati;
- istruzioni di **shift**;
- istruzioni **load** e **store** o istruzioni di trasferimento;
- istruzioni per il **controllo del flusso** o di salto.

Nell'architettura *load* e *store* di ARM, i dati in memoria possono essere caricati esclusivamente nei registri interni alla CPU (*load*), oppure essere trasferiti da un registro in una cella di memoria (*store*).

Per la gestione dei salti condizionati è possibile adottare la soluzione tradizionale, già vista per una tipica architettura 80x86, ma ARM mette a disposizione una *esecuzione condizionale* molto potente, che cambia la prospettiva di programmazione.

### Istruzioni di data processing

Per questo tipo di istruzioni sono previsti:

- indirizzamento con registro;
- indirizzamento immediato (il valore è preceduto da #).

Per esempio:

```
mov r1,#78.
```

Gli operandi possono essere registri o valori immediati (preceduti da #).

È possibile effettuare un'operazione di shift, facendo slittare il secondo operando prima che sia combinato con il primo.

Le istruzioni di data processing comprendono:

- istruzioni aritmetiche (**Tabella 4**);
- istruzioni logiche (**Tabella 5**);
- istruzioni di moltiplicazione (**Tabella 6**) per le quali:
  - vengono memorizzati nel registro di destinazione i 32 bit meno significativi del risultato, mentre gli altri vengono ignorati;
  - il secondo operando non può essere un valore immediato;
  - il registro destinazione non può coincidere con il registro sorgente;
- istruzioni di trasferimento dati tra registri (**Tabella 7**).

**Istruzioni aritmetiche: add, adc, sub, sbc, rsb, rsc**

Istruzione	Significato	Descrizione
add r0,r1,r2	r0 := r1 + r2	Add (somma)
adc r0,r1,r2	r0 := r1 + r2 + C	Add with Carry (somma con Carry)
sub r0,r1,r2	r0 := r1 - r2	Subtract (sottrazione)
sbc r0,r1,r2	r0 := r1 - r2 + C - 1	Subtract with Carry (sottrazione con Carry)
rsb r0,r1,r2	r0 := r2 - r1	Subtract in the reverse order (sottrazione in ordine inverso)
rsc r0,r1,r2	r0 := r2 - r1 + C - 1	Subtract with Carry in the reverse order (sottrazione con Carry in ordine inverso)

† Tab. 4

**Istruzioni logiche: and, orr, eor, bic**

Istruzione	Significato	Descrizione
and r0,r1,r2	r0 := r1 and r2	Logical AND (AND logico)
orr r0,r1,r2	r0 := r1 or r2	Logical (inclusive) OR (OR logico [inclusivo])
eor r0,r1,r2	r0 := r1 xor r2	Logical exclusive OR (OR logico [esclusivo])
bic r0,r1,r2	r0 := r1 and (not r2)	AND logico tra un operando e il complemento del secondo operando

† Tab. 5

**Istruzioni di moltiplicazione: mul, mla**

Istruzione	Significato	Descrizione
mul r0,r1,r2	r0 := r1 * r2	Multiply (moltiplicazione)
mla r0,r1,r2,r3	r0 = (r1 * r2) + r3	Multiply with Accumulate (moltiplicazione con accumulatore)

† Tab. 6

**Istruzioni di trasferimento tra registri: mov, mvn**

Istruzione	Significato	Descrizione
mov r0,r2	r0 := r2	Move (trasferimento)
mvn r0,r2	r0 := not r2	Move Negative (trasferimento negativo)

† Tab. 7

Tutte le istruzioni di data processing possono settare i bit di condizione (N, Z, C, V) del registro di stato CPSR. Per fare ciò occorre aggiungere una s al codice mnemonico dell'istruzione.

Le istruzioni aritmetiche settano tutti i flag (N, Z, C, V).

Le istruzioni logiche e di trasferimento dati settano N e Z.

**Esempio 1**

adds r2,r2,r0	r2 := r2 + r0 (somma) setta N, Z, C, V dal registro di stato CPSR
adcs r3,r3,r1	r3 := r3 + r1 (somma con riporto) setta N, Z, C, V dal registro di stato CPSR

## ■ Istruzioni di shift

Lo shift logico sposta il numero di 1 bit a destra o a sinistra. La [Figura 8](#) mostra il funzionamento dello shift.

L'operazione di shift ([Tabella 8](#)) a destra o a sinistra avviene moltiplicando o dividendo i numeri.

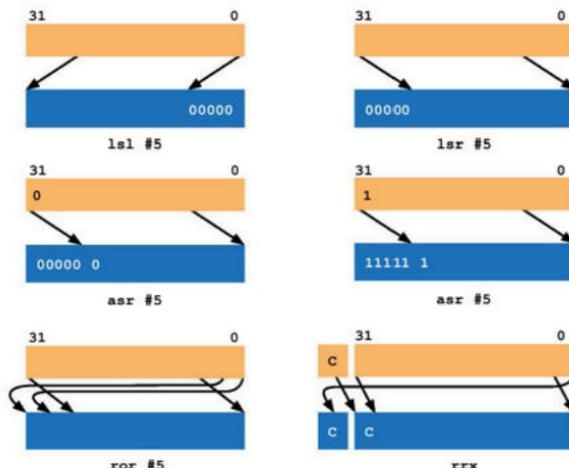
Istruzioni di shift: lsl, lsr, asl, asr, ror, rrx	
Istruzione	Descrizione
lsl	Logical Shift Left (shift logico a sinistra)
lsr	Logical Shift Right (shift logico a destra)
asl	Arithmetic Shift Left (shift aritmetico a sinistra)
asr	Arithmetic Shift Right (shift aritmetico a destra)
ror	Rotate Right (rotazione a destra)
rrx	Rotate Right Extended (rotazione a destra estesa)

◀ Tab. 8

ARM non ha istruzioni di shift che vengono eseguite da sole, ma le implementa «shiftando» il secondo operando di istruzioni aritmetiche o logiche.

### Esempio 2

Istruzione	Significato
add r3,r2,r1, lsl #3	$r3 := r2 + 8 \times r1$
add r5,r5,r3, lsl r2	$r5 := r5 + 2^{r2} \times r3$



◀ Fig. 8 Il funzionamento dell'istruzione di shift.

### ■ notabene

Le istruzioni di store hanno la stessa sintassi delle istruzioni di *load*.

### ■ Istruzioni di load e store

Sono le istruzioni che accedono alla memoria: **load** per leggere da memoria in un registro, **store** per scrivere un dato da un registro in memoria. ARM offre i seguenti metodi di indirizzamento:

- Indirizzamento indiretto (*Indirect Addressing*).
- Indirizzamento base+offset.
- Indirizzamento pre-indicizzato (*Pre-Indexed Addressing*).
- Indirizzamento post-indicizzato (*Post-Indexed Addressing*).
- Indirizzamento multiplo.

Vengono utilizzati indirizzi relativi costituiti da base e spiazzamento.

Il trasferimento può essere:

- di registri singoli, per il trasferimento di un dato (byte, half-word, word) tra un registro e la memoria;
- di registri multipli, per il trasferimento di grandi quantità di dati (per la chiamata e il ritorno da funzione, per salvare/ripristinare lo stato dei registri e per copiare blocchi di dati).

Trasferimento di registri singoli			
Istruzione	Significato	Descrizione	Indirizzamento
ldr r0,[r1]	r0 := mem <sub>32</sub> [r1]	Trasferisce in r0 un dato a 32 bit dalla locazione di memoria il cui indirizzo è in r1	Indiretto
ldrb r0,[r1]	r0 := mem <sub>8</sub> [r1]	Trasferisce in r0 un dato a 8 bit dalla locazione di memoria il cui indirizzo è in r1	Indiretto
str r0,[r1]	mem <sub>32</sub> [r1] := r0	Trasferisce nei 4 byte di memoria, il cui indirizzo è in r1, il contenuto di r0	Indiretto
ldr r0,[r1,#4]	r0 := mem <sub>32</sub> [r1 + 4]	Trasferisce in r0 il contenuto della locazione il cui indirizzo è in r1 aumentato di 4 (r1 è la base, 4 lo spiazzamento)	Base + offset (spiazzamento)
ldr r0,[r1,#4]!	r0 := mem <sub>32</sub> [r1 + 4] r1 := r1 + 4	Trasferisce in r0 il contenuto della locazione il cui indirizzo è in r1 aumentato di 4 (r1 è la base, 4 lo spiazzamento); il registro base viene modificato ed è usato come registro indice	Pre-indicizzato
ldr r0,[r1],#4	r0 := mem <sub>32</sub> [r1] r1 := r1 + 4	Trasferisce in r0 il contenuto della locazione il cui indirizzo è in r1; il registro base viene modificato ed è usato come registro indice	Post-indicizzato

↑ Tab. 9

Si noti che le istruzioni ldr/str sono della forma ldr<suffix>:

ldr      trasferimento di una **word** (32 bit)

ldrb     trasferimento di un **byte** (8 bit)

ldrh     trasferimento di **mezza word** (16 bit).

### Trasferimento di registri multipli

Istruzione	Significato	Descrizione	Indirizzamento
ldmia r1,{r0,r2,r3}	r0 := mem <sub>32</sub> [r1] r2 := mem <sub>32</sub> [r1 + 4] r3 := mem <sub>32</sub> [r1 + 8]	Load Multiple Register (carica più registri da posizioni di memoria consecutive utilizzando un registro base, con incremento)	Increment After (Post-indicizzato)
ldmib r1,{r0,r2,r5}	r0 := mem <sub>32</sub> [r1 + 4] r2 := mem <sub>32</sub> [r1 + 8] r5 := mem <sub>32</sub> [r1 + 12]	Load Multiple Register (carica più registri da posizioni di memoria consecutive utilizzando un registro base, con incremento)	Increment Before (Pre-indicizzato)
ldmda r1,{r0,r2,r6}	r0 := mem <sub>32</sub> [r1] r2 := mem <sub>32</sub> [r1 - 4] r6 := mem <sub>32</sub> [r1 - 8]	Load Multiple Register (carica più registri da posizioni di memoria consecutive utilizzando un registro base, con decremento)	Decrement After (Post-indicizzato)
ldmdb r1,{r0,r2,r5}	r0 := mem <sub>32</sub> [r1 - 4] r2 := mem <sub>32</sub> [r1 - 8] r5 := mem [r1 - 12]	Load Multiple Register (carica più registri da posizioni di memoria consecutive utilizzando un registro base, con decremento)	Decrement Before (Post-indicizzato)

↑ Tab. 10

È possibile effettuare un trasferimento di registri multipli in cui, nell'istruzione, i registri sono elencati in ordine casuale, e un range di registri può essere indicato (ad esempio) con r1-r5. In generale le istruzioni di trasferimento di registri multipli hanno la funzione di copiare un blocco di dati da registri a memoria (o viceversa).

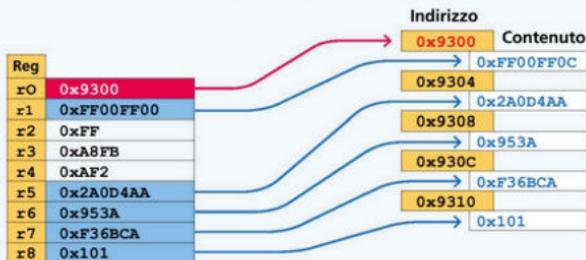
#### notabene

Le istruzioni di trasferimento a blocchi sono utilizzate per caricare (**LDM** - Load Multiple registers) o archiviare (**STM** - Store Multiple registers) blocchi di dati dalla/alla memoria appoggiandosi ai registri.

### Esempio 3

```
stm r0,{r1, r5-r8}
```

Copia il contenuto dei registri r1, r5, r6, r7, r8 in memoria a partire dall'indirizzo contenuto in r0, come illustrato nella figura.



### ■ Istruzioni per il controllo del flusso

Il controllo del flusso è realizzato con istruzioni che testano i 4 bit (NZCV) di condizione del registro di stato (CPSR) per verificare una determinata condizione: si tratta delle istruzioni di salto e delle istruzioni condizionate.

Per le istruzioni di salto è utilizzato l'indirizzamento immediato.

I bit del registro di stato possono essere modificati o da istruzioni di confronto o da una qualsiasi istruzione a cui è stato aggiunto il suffisso «s» al codice mnemonico.

### Esempio 4

adrs r0,r1,r2

esegue  $r0 = r1 + r2$  e modifica i bit del registro di stato.

L'istruzione add senza la «s» (che può essere usata per quasi tutte le istruzioni) non ha effetto sul registro di stato.

### Istruzioni di confronto

Modificano solo i 4 bit del registro di stato.

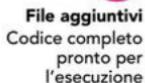
Istruzioni di confronto: cmp, cmn, tst, teq	
Istruzione	Descrizione
cmp r1,r2	Modifica il registro dei Flag dopo aver effettuato $r1 - r2$
cmn r1,r2	Modifica il registro dei Flag dopo aver effettuato $r1 + r2$
tst r1,r2	Modifica il registro dei Flag dopo aver effettuato $r1 \text{ and } r2$
teq r1,r2	Modifica il registro dei Flag dopo aver effettuato $r1 \text{ xor } r2$

Per esempio, il codice per eseguire 50 volte un ciclo potrebbe essere scritto nel modo seguente:

```

mov r0,#1
loop:
    add r0,r0,#1
    cmp r0,#50
    ble loop
o in modo più compatto
mov r0,#50
loop:
    subs r0,r0,#1
    bne loop

```



### File aggiuntivi

Codice completo pronto per l'esecuzione

Nella prima soluzione i bit del registro di stato sono stati modificati dall'istruzione cmp; nella seconda soluzione, invece, dall'istruzione subs.

### Istruzioni di salto

Salto incondizionato	
b <etichetta>	Salta all'istruzione posta all'indirizzo associato all'etichetta etichetta
Salto condizionato	
Le istruzioni sono della forma b seguita da una condizione. Esempio: beq, bne La Tab. 11 mostra tutte le possibili condizioni. La condizione dipende da 4 bit (Z, N, C, V) del registro di stato CPSR e viene specificata utilizzando un suffisso di due lettere.	

Suffisso	Significato	Flag testati
eq	Equal	Z=1
ne	Not Equal	Z=0
vs	Overflow Set	V=1
vc	Overflow Clear	V=0
al	Always	any
nv	Never	
hi	Higher	C=1 AND Z=0
ls	Lower than or Same	C=0 AND Z=1
pl	Plus Clear	N=0
mi	Minus Set	N=1
cs/hs	Carry Set	C=1
cc/lo	Carry Clear	C=0
ge	Greater than or Equal	N=1,V=1 OR N=0,V=0 -> N==V
lt	Less than	N=1,V=0 OR N=0,V=1 -> N!=V
gt	Greater than	(N=1,V=1 OR N=0,V=0) AND Z=0 -> !Z AND (N==V)
le	Less than or Equal	(N=1,V=0 OR N=0,V=1) OR Z=1 -> Z OR (N!=V)

### Istruzioni condizionate

Una particolare caratteristica di ARM è che le istruzioni possono essere eseguite in modo condizionato (**esecuzione condizionale**), aggiungendo i suffissi condizionali al codice mnemonic delle istruzioni. Ciò rende il codice molto più compatto. Tali istruzioni testano i bit del registro di stato e vengono eseguite solo se la condizione espresso nelle istruzioni è vera.

I primi 4 bit (i più significativi) di un'istruzione, infatti, contengono una condizione e l'istruzione viene eseguita solo se la condizione è quella espressa da questi 4 bit. La **configurazione 1110** indica che l'**istruzione è incondizionata**. Queste istruzioni possono essere combinate con il suffisso «s» (posto dopo il suffisso di condizione).

Tab 11. Condizioni utilizzabili nelle istruzioni di salto.

### Esempio 5

```
movs r0,r1 ; trasferisce il contenuto di r1 in r0
              ; e setta il registro di stato
moveq r0,#1 ; se Z=1 trasferisce 1 in r0; infatti
              ; la condizione EQ è vera se Z=1; tale
              ; bit viene settato da MOVS se r0=0.
```

### notabene

Il formato di un'istruzione ARM si trova nell'Approfondimento.

### Esempio 6

```
cmp r5,r6 ; confronta r5 e r6
addne r5,r5,r6 ; se sono diversi, li somma
                  ; e pone il risultato in r5
```

**Pit Stop**

- 5 Che cosa sono le istruzioni condizionate?
- 6 Quali sono le istruzioni di salto?
- 7 Che differenza c'è tra salto condizionato e salto incondizionato?

**Esempio 7**

```

subs r0,r1,r2 ; esegue la sottrazione r1-r2
                ; e modifica il registro di stato
addeq r2,r2,#1 ; se subs ha dato risultato 0
                  ; (r1=r2), somma 1 a r2 e pone
                  ; il risultato in r2
beq label      ; se subs ha dato risultato 0,
                  ; salta a label

```

**Esempio 8**

In questo esempio mettiamo a confronto un algoritmo per il calcolo del MCD effettuato in modo tradizionale e un algoritmo per il calcolo del MCD che utilizza istruzioni condizionate. Notiamo come in questo secondo caso il codice è decisamente più semplice e compatto.

Metodo tradizionale	Metodo con istruzioni condizionate
MCD: cmp r0,r1 beq FINE blt MIN sub r0,r0,r1 b MCD MIN: sub r1,r1,r0 b MCD FINE:	MCD: cmp r0,r1 subgt r0,r0,r1 sublt r1,r1,r0 bne MCD

**File aggiuntivi**

Codice completo pronto per l'esecuzione

**8 Come si implementano le strutture di controllo in ARM**

Vediamo come le strutture di controllo, tipiche dei linguaggi di alto livello, possono essere implementate in ARM.

**Selezione (if)**

Pseudocodifica	ARM – soluzione tradizionale (supponiamo che r0=a e r1=b)	ARM – soluzione con istruzioni condizionate (più compatta) (supponiamo che r0=a e r1=b)
if (a == b) then a = a + 1 else a = b fine if	cmp r0,r1 beq then mov r0,r1 b fine then: add r0,r0,#1 fine:	NOTA BENE: inizializzare opportunamente r0 e r1 cmp r0,r1 addeq r0,r0,#1 movne r0,r1

**Ciclo (while)**

Pseudocodifica	ARM – soluzione tradizionale (supponiamo che r0=i e r1=j)	ARM – soluzione con istruzioni condizionate (più compatta) (supponiamo che r0=i e r1=j) (SCAMBIO DI VALORI tra r0 e r1)
while (i != 0) do i = i - 1 j = j + 1 fine while	while: cmp r0,#0 beq fine sub r0,r0,#1 add r1,r1,#1 b while fine:	NOTA BENE: inizializzare opportunamente r0 e r1 while: cmp r0,#0 subne r0,r0,#1 addne r1,r1,#1 bne while

## 9 Vettori

Con ARM è possibile manipolare vettori (array), cioè strutture di dati dello stesso tipo.

Questi sono allocati in memoria in locazioni contigue; si accede all'intero vettore utilizzando l'indirizzo del primo elemento (base), mentre si accede agli altri elementi usando un indice che ha la funzione di spiazzamento rispetto alla base. Le istruzioni per leggere e scrivere elementi di un vettore sono ldr/str.

### Esempio 9

Scriviamo in r1 la somma dei primi tre elementi del vettore (di interi) il cui indirizzo base è contenuto in r0.

```
ldr r1,[r0]
ldr r2,[r0,#4]
add r1,r1,r2
ldr r2,[r0,#8]
add r1,r1,r2
```



**File aggiuntivi**  
Codice completo pronto per l'esecuzione

### Pit Stop

- 8 Come vengono letti e scritti i vettori con le istruzioni ARM?

## 10 Etichette e direttive

### Etichette

Le etichette (label) consentono di associare un nome a un indirizzo di memoria e servono per identificare una porzione di codice, senza necessariamente conoscere l'indirizzo di memoria in cui il codice è allocato. Sarà poi compito dell'assembler tradurre le etichette nei corrispondenti effettivi indirizzi.

### Direttive

Tutte le direttive iniziano con un «.».

Le **direttive per la struttura di un programma** sono quelle elencate nella tabella seguente:

Direttiva	Significato
.text	Indica l'inizio del codice
.data	Indica che ciò che segue sono dati
.global	Rende globale la visibilità del codice
.end	Indica la fine del codice

Esistono poi delle **direttive** utili per il **caricamento dei dati in memoria**, mostrate nella tabella seguente:

Direttiva	Significato	Esempio
.word n	Alloca spazio per un numero di 4 byte e gli assegna il valore specificato	.word 38 .word 0xF5ABBB15
.ascii "stringa"	Carica il codice espresso tra apici	.ascii "questa è una stringa"
.asciz	Aggiunge alla stringa uno 0 come carattere di fine stringa	.asciz "questa stringa finisce con 0"

Le **costanti** sono definite con la direttiva `.equ`, che deve essere posta nella sezione `.data` e associa un nome a un valore. Nel programma può essere utilizzato il nome per migliorare la leggibilità del codice. Per esempio:

```
.data
...
.equ incremento,1
.equ var1=100
```

È possibile infine assegnare nomi simbolici ai registri con la direttiva `.req` sempre al fine di migliorare la leggibilità. In genere si associano dei nomi che identificano la funzione di un registro.

## 11 Le eccezioni

In ambiente ARM per **eccezione** si intende l'**interruzione** dell'esecuzione di un programma (in esecuzione in modalità utente, non privilegiata) a favore di un altro programma (in esecuzione in un'altra modalità utente, privilegiata). Come sappiamo, questo può avvenire a causa di interrupt (hardware o software) o di chiamate al sistema operativo (system call).

### ■ Tipi di eccezioni

ARM gestisce le seguenti eccezioni (**Tabella 12**):

- interruzioni hardware esterne: IRQ, FIQ;
- interruzioni software interne (trap): Abort (ABT), Undefined (UNDEF);
- interruzioni software interne (system call): Supervisor (SVC), System (SYS).

Quando si verifica un'eccezione, il processore cambia modalità di funzionamento e viene modificato opportunamente il registro di stato (vedi la **Figura 5**).

La **Tabella 12** riassume, per ogni modalità di funzionamento (eccezione), il valore dei bit del registro di stato e i relativi registri.

→ **Tab. 12** Configurazione dei bit del registro di stato per ogni tipo di eccezione (uc = unchanged).

	I	F	T	MODE					
	7	6	5	4	3	2	1	0	
Abort	1	1	0	1	0	1	1	1	
FIQ	1	1	0	1	0	0	0	1	
IRQ	1	uc	0	1	0	0	1	0	
Supervisor	1	uc	0	1	0	0	1	1	
System	1	1	0	1	1	1	1	1	
Undefined	1	uc	0	1	1	0	1	1	
User	0	0	0	1	0	0	0	0	

↓ **Tab. 13** A ogni eccezione sono associati dei registri.

M[4;0]	Mode	Registri accessibili
0b10000	User	PC, da r14 a r0, CPSR
0b10001	FIQ	PC, da r14_fiq a r8_fiq, da r7 a r0, CPSR, SPSR_fiq
0b10010	IRQ	PC, r14_irq, r13_irq, da r12 a r0, CPSR, SPSR_irq
0b10011	Supervisor	PC, r14_svc, r13_svc, da r12 a r0, CPSR, SPSR_svc
0b10111	Abort	PC, r14_abt, r13_abt, da r12 a r0, CPSR, SPSR_abt
0b11011	Undefined	PC, r14_und, r13_und, da r12 a r0, CPSR, SPSR_und
0b11111	System	PC, da r14 a r0, CPSR (ARMv4 e successive)

## Gestione delle eccezioni

In ogni caso è necessario effettuare il cambio di contesto e l'identificazione dell'eccezione.

In particolare:

- Viene salvato lo stato del processo in corso copiando il contenuto del Program Counter e del registro di stato rispettivamente in r14 e in SPSR, associati all'eccezione da gestire.
- Il processore passa dalla modalità di funzionamento *User* alla modalità associata all'eccezione da gestire.
- In caso di *Reset* o *FIQ* vengono disabilitati gli interrupt (settando i bit del registro di stato).
- Viene memorizzato l'indirizzo della prossima istruzione (indirizzo di ritorno) nel registro LR associato all'eccezione.
- Al Program Counter è assegnato l'indirizzo di inizio della routine che deve essere eseguita per «servire» l'eccezione (**Tabella 14**).
- Viene eseguita la routine di gestione dell'eccezione.

Al termine dell'esecuzione della routine associata all'eccezione deve essere ripristinato lo stato del processo (se previsto dall'eccezione):

- viene copiato SPSR in CPSR (ripristino del registro di stato);
- viene copiato LR in PC (ripristino del Program Counter);
- viene ripristinata la modalità *User*.

### notabene

Il vettore delle eccezioni contiene gli indirizzi di inizio delle routine associate alle eccezioni.

◀ **Tab. 14** Vettore delle eccezioni.

Indirizzo	Descrizione	Modo
0x00000000	Reset	SVC
0x00000004	Undefined Instruction (UDEF)	UDEF
0x00000008	Software Interrupt (SWI)	SVC
0x0000000C	Prefetch Abort (PABT)	ABT
0x00000010	Data Abort (DABT)	ABT
0x00000014	RISERVATO	-
0x00000018	Interrupt (IRQ)	IRQ
0x0000001C	Fast Interrupt (FIQ)	FIQ

Come esempio di gestione di un'eccezione consideriamo l'**eccezione Reset** e vediamo che cosa accade quando si verifica un Reset:

- il processore passa alla modalità Supervisor;
- vengono disabilitati gli interrupt (I = 1, F = 1 nel registro di stato);
- se il processore era in modalità Thumb, viene riportato in modalità ARM (T = 0 nel registro di stato);
- PC = 0x00;
- non viene eseguito il ritorno; Reset è non rientrante.

### Pit Stop

- 9 Cosa sono le eccezioni nell'architettura ARM?
- 10 Quali sono i tipi di eccezioni gestiti da ARM?
- 11 Come viene modificato il registro di stato quando si verifica un'eccezione?

## ■ Priorità delle eccezioni

Quando si verificano due eccezioni contemporaneamente, queste vengono servite in base alla priorità descritta nella **Tavella 15**:

Eccezione	Reset	Data Abort	FIQ	IRQ	Prefetch Abort	Undefined instruction, SWI
Priorità	1 (maggiore)	2	3	4	5	6 (minore)

↑ Tab. 15 Priorità delle eccezioni. Reset ha priorità maggiore.

## 12 Interrupt software (SWI)

Gli *interrupt software* sono delle *Supervisor Call*, cioè chiamate al sistema operativo che interrompono l'esecuzione del programma in corso, a favore di un particolare servizio del sistema operativo. Ciò significa che la modalità del processore passa a *Supervisor*, il CPSR viene salvato in SPSR e il programma salta all'esecuzione della routine richiamata.

Per accedere ai servizi del sistema operativo, esattamente come succede nei sistemi della famiglia 80x86 con l'istruzione INT, occorre servirsi dell'istruzione **swi (Software Interrupt)**, che può accettare argomenti e restituire valori.

Spesso si utilizza l'istruzione **swi 0** preceduta dal caricamento nel registro r7 del numero che identifica il servizio da eseguire.

Per esempio, per chiudere un programma e passare il controllo al sistema operativo si scrive:

```
mov r7,#1
swi 0
```

A volte potrebbe essere necessario passare dei parametri alla routine da eseguire; in questo caso si utilizzano dei registri specifici.

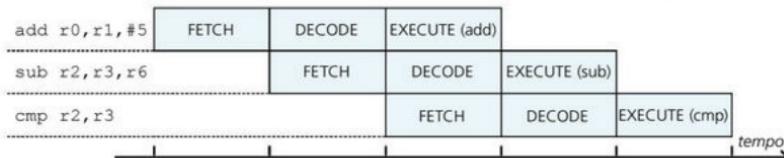
## 13 La pipeline e le istruzioni

La versione v3 di ARM presenta una semplice pipeline a 3 stadi: *fetch*, *decode*, *execute*.

Per esempio le istruzioni:

```
add r0,r1,#5
sub r2,r3,r6
cmp r2,r3
```

che non effettuano accesso alla memoria, richiedono una pipeline a tre stadi con conseguente tempo di esecuzione, come mostra il diagramma seguente.



Per le istruzioni di *load* e *store*, che accedono alla memoria, occorre introdurre uno stallo perché, prima dell'execute, occorre calcolare l'indirizzo con conseguente aumento del tempo di esecuzione.

Le versioni successive di ARM hanno notevolmente migliorato le prestazioni, con pipeline a 5 stadi (simili a quelle del processore MIPS) fino ad arrivare ad architetture con pipeline a 6 stadi o più.

# Approfondimento - Tecnologia

## Formato di una generica istruzione ARM

Il formato di una generica istruzione è rappresentato nella **Figura 1**. Il significato è mostrato in **Figura 2**.

3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9 8	8 7	7 6	5 4	4 3	3 2	2 1	1 0	Instruction Type	
Condition	0	0	1	OPCODE				S	Rn		Rs		OPERAND-2										Data processing								
Condition	0	0	0	0	0	0	A	S	Rd		Rn		Rs		1	0	0	1	Rm		Multiply										
Condition	0	0	0	0	1	U	A	S	Rd HIGH		Rd LOW		Rs		1	0	0	1	Rm		Long Multiply										
Condition	0	0	0	1	0	B	0	0	Rn		Rd		0 0 0 0 1		0	0	0	1	Rm		Swap										
Condition	0	1	I	P	U	B	W	L	Rn		Rd		OFFSET										Load/Store-Byte/Word								
Condition	1	0	0	P	U	B	W	L	Rn		REGISTER LIST										Load/Store Multiple										
Condition	0	0	0	P	U	1	W	L	Rn		Rd		OFFSET 1		1	S	H	1	OFFSET 2		Halfword Transfer Imm Off										
Condition	0	0	0	P	U	0	W	L	Rn		Rd		0 0 0 0 1		S	H	1	Rm		Halfword Transfer Reg Off											
Condition	1	0	1	L	BRANCH OFFSET															Branch											
Condition	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn		Branch Exchange					
Condition	1	1	0	P	U	N	W	L	Rn		CRd		CPNum		OFFSET										Coprocessore data transfer						
Condition	1	1	1	0	Op-1				CRn		CRd		CPNum		OP-2	0	CRm		Coprocessore data operation												
Condition					Op-1		L	CRn		Rd		CPNum		OP-2	1	CRm		Coprocessore REG transfer													
Condition	1	1	1	1	SWI NUMBER															Software Interrupt											

↑ Fig. 1 Formato di una generica istruzione ARM.

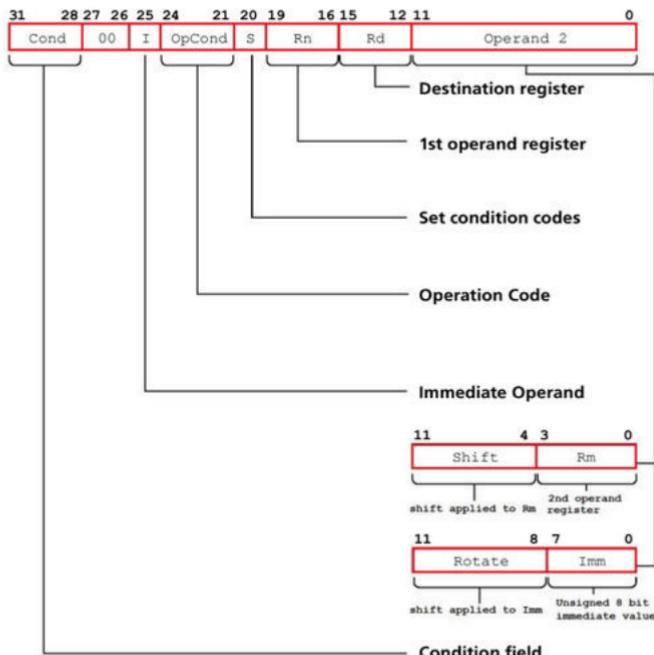


Fig. 2 Significato dei 32 bit di una generica istruzione ARM.

Fig. 3 Significato dei bit di condizione in un'istruzione.

I primi 4 bit che esprimono la condizione possono essere combinati come in Figura 3:

The Condition Field																Instruction Type																		
3	3	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0						
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0			
Condition	0	0	1	OPCODE	S	Rn	Rs	OPERAND-2										Data processing																
0000 = EQ - Z set (equal)																1001 = LS - C clear or Z (set unsigned lower or same)																		
0001 = NE - Z (not equal)																1010 = GE - N set and V set, or N clear and V clear (> or =)																		
0010 = HS / CS - C set (unsigned higher or same)																1011 = LT - N set and V clear, or N clear and V set (>)																		
0011 = LO / CC - C clear (unsigned lower)																1100 = GT - Z clear, and either N set and V set, or N clear and V set (>)																		
0100 = MI - N set (negative)																1101 = LE - Z set, or N set and V clear, or N clear and V set (<, or =)																		
0101 = PL - N clear (positive or zero)																1110 = VS - V set (overflow)																		
0110 = VC - V clear (no overflow)																1111 = NV - reserved																		
1000 = HI - C set and Z clear (unsigned higher)																																		

## Esempio 1

Consideriamo questo stralcio di programma:

```
01 20 82 e2      add r2,r2,#1
02 20 82 e2      add r2,r2,#2
```

Consideriamo l'istruzione `add r2,r2,#1`.

È un'istruzione di data processing il cui codice di 4 byte si interpreta nel seguente modo (ricordiamo che il formato è little endian e, quindi, va letto al contrario):

```
e2 82 20 01
```

In binario:

```
1110 00 1 0100 0 0010 0010 0000 0000 0001 operando 2 (#1)
1110 00 1 0100 0 0010 0010 0000 0000 0001 Rn Rd (r2,r2)
1110 00 1 0100 0 0010 0010 0000 0000 0001 S (0)
1110 00 1 0100 0 0010 0010 0000 0000 0001 OpCode (ADD = 0100)
1110 00 1 0100 0 0010 0010 0000 0000 0001 Immediate (Immediate 1)
1110 00 1 0100 0 0010 0010 0000 0000 0001 00 fissi
1110 00 1 0100 0 0010 0010 0000 0000 0001 Condizione (Senza Condizione)
```

L'istruzione `add r2,r2,#2` ha codice operativo:

```
e2 82 20 02
```

In binario:

```
1110 00 1 0100 0 0010 0010 0000 0000 0002 operando 2 (#2)
... (come sopra)
```

# Laboratorio

## Programmazione in Assembly ARM

Per poter scrivere ed eseguire programmi scritti in Assembly ARM, proponiamo due soluzioni:

- utilizzare l'emulatore VisUAL;
- utilizzare un ambiente reale ARM, per esempio su Raspberry Pi.

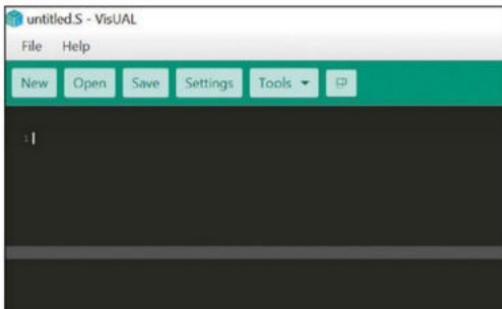
Di seguito presentiamo i due ambienti e, successivamente, alcuni problemi di programmazione risolti in ARM.

### ATTIVITÀ 1

#### Utilizzare un emulatore ARM

##### Passo 1: installare e aprire VisUAL

Selezionare *New* per aprire l'editor e scrivere un nuovo file sorgente.



##### Passo 2: scrivere il codice

Scrivere il codice sorgente e salvare il file sorgente. Per esempio:

```

New Open Save Settings

1      mov    r1, #45
2      mov    r2, #3
3      mov    r11, #4
4      add    r3, r2, r11
5
6

```

#### notabene

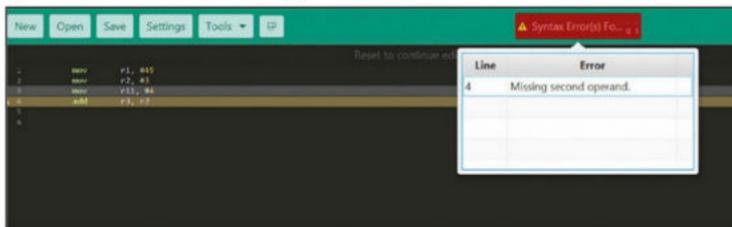
VisUAL è un emulatore che consente di scrivere programmi in Assembly ARM ed effettuarne il debug con semplicità. È stato sviluppato da Salman Arif dell'Imperial College, Londra. VisUAL2 è una versione successiva.

### Passo 3: assemblare ed eseguire

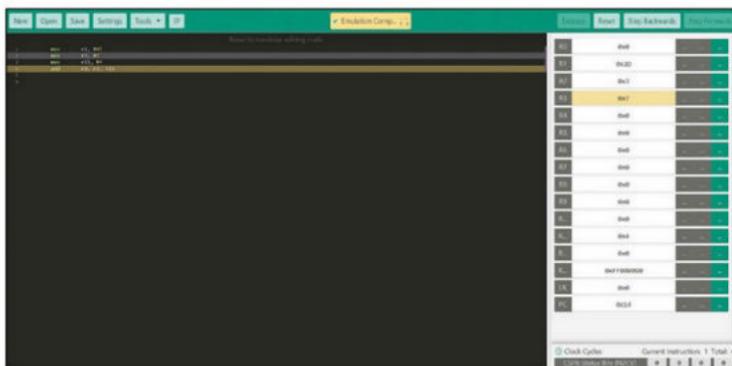
Selezionare *Execute* per assemblare ed eseguire il programma.

In presenza di errori, essi saranno segnalati, altrimenti il programma sarà eseguito.

Di seguito mostriamo un esempio di rilevazione di un errore dovuto alla mancanza di un operando nell'istruzione *add* alla riga 4.



È possibile verificare la corretta esecuzione del programma osservando il contenuto dei registri.



## ATTIVITÀ 2

### Impariamo a leggere il contenuto della memoria e dei registri della CPU

Consideriamo il seguente programma, che legge due numeri dalla memoria, li somma e scrive il risultato nella memoria.

#### Scenario

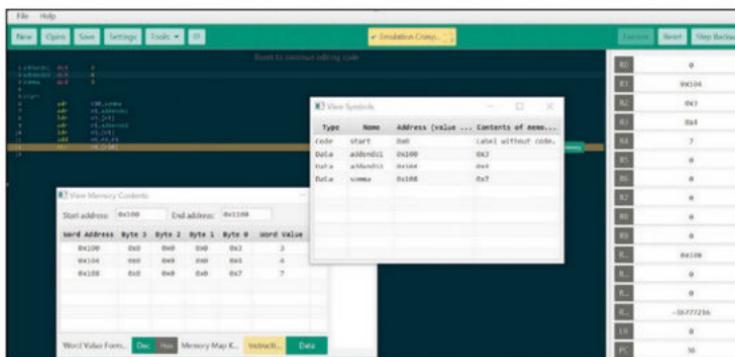
Siano *addendo1* e *addendo2* le due variabili da sommare.

#### Obiettivo

La variabile *somma* deve contenere la somma di *addendo1* + *addendo0*.

#### Svolgimento

Scriviamo il codice sorgente ed eseguiamo il programma.



Al termine dell'esecuzione, selezionando **Tools**, apriamo le finestre per visualizzare la *symbol table* e il contenuto della memoria e verificare il funzionamento del programma. Il funzionamento è verificato osservando che la variabile somma contiene il valore 7, che è la somma di addendos e addendo2.

### ATTIVITÀ 3

#### Utilizzare processore ARM su Raspberry Pi

In questa attività mostriamo come scrivere, assemblare, eseguire e fare il debug di un programma su Raspberry Pi.

Per lavorare, occorre collegarsi al Raspberry da remoto o mediante un monitor, una tastiera e un mouse.

##### Passo 1: creazione del file sorgente (hello.s)

Scriviamo il file sorgente e salviamolo con il nome hello.s

- Tramite un editor di testo, per esempio **nano**, scriviamo il codice sorgente.  
Osserviamo che ogni file sorgente deve avere un punto di partenza che, per impostazione predefinita nell'assemblatore, corrisponde all'etichetta `_start`:
- Salviamo il file con il nome hello.s

```
@scrive hello
.global _start
_start:
    mov r7, #4
    mov r0, #1
    mov r2, #8
    ldr r1, =string
    swi 0

    _exit:
    mov r0, #0
    mov r7, #1
    swi 0
```

```
.data
string:
.ascii "hello"
crlf:
.byte 13,10,0
```

**Passo 2: lancio dell'assemblatore e generazione del file oggetto (hello.o)**

Mandiamo in esecuzione il programma Assembler AS per compilare il sorgente e generiamo il file oggetto hello.o

```
$ as -g -o hello.o hello.s
```

L'opzione `-o` indica il nome del file oggetto (con estensione `.o`) prodotto dall'assembler. In caso di errore di sintassi verrà emessa una segnalazione con la descrizione dell'errore e il numero della riga dell'errore.

L'opzione `-g` non è obbligatoria, ma è necessaria per la produzione dei riferimenti necessari per l'utilizzo del programma di debug (gdb).

**Passo 3: linker e generazione del file eseguibile (hello)**

Mandiamo in esecuzione il linker per generare il file eseguibile hello.

Il comando ld applica la procedura di «linker» per ottenere il file eseguibile.

```
$ ld -o hello hello.o
```

**Passo 4: esecuzione del programma**

Per eseguire il programma scriviamo:

```
$ ./hello
```

**ATTIVITÀ 4****Screenshot relativo ai passi descritti nell'attività 3**

Listato del file hello.s

Chiamata di assemblador ed linker ed esecuzione.

Si noti la label `_start`: che determina il punto di partenza del programma e l'utilizzo delle istruzioni swi, con parametri diversi, per richiamare la stampa a video della stringa «HELLO!» e per la chiusura corretta del programma.

```
File Edit Iabs Help
pi@raspberrypi ~/Desktop/arm1 $ cat hello.s
.global _start
_start:
    MOV R7, #4
    MOV R0, #1
    MOV R2, #8
    LDR R1, =string
    SWI 0          /* System call */

_exit:
    MOV R0,#0      /* exit code */
    MOV R7, #1      /* parameter for exit */
    SWI 0          /* System call for Linux */

.data
string:
.ascii "HELLO!"
CRLF:
.byte 13,10,0
pi@raspberrypi ~/Desktop/arm1 $ as -g -o hello.o hello.s
pi@raspberrypi ~/Desktop/arm1 $ ld -o hello hello.o
pi@raspberrypi ~/Desktop/arm1 $ ./hello
HELLO!
```

## ATTIVITÀ 5

### Utilizzo del Debugger gdb (GNU Debugger)

#### Passo 1: attivare il debug

L'uso di gdb (GNU Debugger) permette di controllare l'esecuzione passo passo delle istruzioni e di inserire dei breakpoint per poter osservare il programma in esecuzione e visionare il contenuto dei registri e della memoria.

Per attivare il debug scriviamo:

```
$gdb ./hello
```

Possiamo capire che siamo in modalità debug dal prompt gdb.

#### Passo 2: comandi disponibili

Digitando help otteniamo la lista dei comandi disponibili.

Lo screenshot che segue mostra l'attivazione del debugger e l'help dei comandi, i principali dei quali sono elencati nella Tabella 1.

```
pi@raspberrypi:~/Desktop/arm1$ gdb ./hello
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY; to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "arm-linux-gnueabihf".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/pi/Desktop/arm1/hello...done.
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
```

Comando	Azione
help h	Aiuto sui comandi gdb.
quit q	Esce da gdb.
list l l num _ riga	Mostra le 10 righe successive del programma sorgente (list- mostra le 10 righe precedenti del programma sorgente)
disas disas label /r disas addr1,addr2	Disasembela l'istruzione della label, oppure dall'indirizzo addr1 all'indirizzo addr2.
run r	Esegue il programma fino al primo breakpoint (se impostato).
info registers info variables info address variabile info break	Ritorna lo stato dei registri e delle variabili nome e i breakpoint impostati.
breakpoint (oppure b) num _ riga/label:	Inserisce un breakpoint alla riga indicata o alla label indicata (per esempio: b start).

Comando	Azione
breakpoint (oppure c)	Riprende l'esecuzione del programma.
stepi	Esegue (step) l'istruzione successiva.
next n	Esegue la linea di codice successiva incluse le funzioni.
delete n d n	Cancella il breakpoint numero <i>n</i> .
x [/Nuf ] addr	Examine memory: esamina l'indirizzo di memoria <i>addr</i> . Opzionali: N numero di unità da mostrare; u size (byte, due, quattro, otto byte) f formato di stampa

Tab. 1 Comandi per il debug.

### Passo 3: esempi di utilizzo del debugger

Dopo aver attivato il debugger con:

\$ gdb ./hello

proviamo alcuni comandi.

```
(gdb) list 5
1 .global _start
2
3     .text:
4         MOV R7, #4
5         MOV R0, #1
6         MOV R2, #0
7         LDR R1, =string
8         SWI 0          /* System call */
9
10    .exit:
(gdb) disas _start
Dump of assembler code for function _start:
0x00000074 <_start>:   mov    r7, #4
0x00000078 <+4>:   mov    r0, #1
0x0000007c <+8>:   mov    r2, #0
0x00000080 <+12>:  ldr    r1, [pc, #12]  : 0x0094 <_exit+12>
0x00000084 <+16>:  svc    0x00000000
End of assembler dump.
(gdb) run
Starting program: /home/pi/Desktop/arm1/hello
HELLO!
[Inferior 1 (process 8692) exited normally]
(gdb) b _exit
Breakpoint 1, _exit () at hello.s:12.
(gdb) run
Starting program: /home/pi/Desktop/arm1/hello
HELLO!
```

list 5  
visualizza il codice del programma prima e dopo la riga 5;

disas \_start  
disassembela e mostra le istruzioni del programma in esadecimale;

run  
esegue il programma;

b \_exit  
inserisce un breakpoint alla label \_exit; se lanciamo nuovamente il programma osserviamo che si ferma alla linea 12 e mostra l'istruzione relativa;

```
Breakpoint 1, _exit () at hello.s:12
12     MOV R7, #1          /* parameter for exit */
(gdb) disas /r
Dump of assembler code for function _exit:
0x0000008000 <_exit>:   00 00 00 e3    mov    r0, #0
=> 0x0000008004 <+4>:   01 70 00 e3    mov    r7, #1
0x0000008008 <+8>:   00 00 00 e1    svc    0x00000000
0x000000800c <+12>:  98 00 01 00    muleq  r1, r8, r0
End of assembler dump.
(gdb) stepi
13     SWI 0          /* System call for Linux */
(gdb) stepi
[Inferior 1 (process 8697) exited normally]
(gdb) 
```

disas  
disassembela l'istruzione precedente e mostra il codice delle istruzioni in esadecimale;

stepi  
eseguiamo l'istruzione (step) successiva (stepi).

Il seguente screenshot mostra il contenuto dei registri della CPU prima e dopo l'esecuzione dell'istruzione `mov r2,#8`.

```

Breakpoint 2, _start () at hello.elf:4
4      MOV R0, #1
(gdb) info r
r0      0x0      0
r1      0x0      0
r2      0x0      0
r3      0x0      0
r4      0x0      0
r5      0x0      0
r6      0x0      0
r7      0x4      4
r8      0x0      0
r9      0x0      0
r10     0x0      0
r11     0x0      0
r12     0x0      0
sp      0x7efff690    0x7efff690
lr      0x0      0
pc      0x8078  0x8078 <_start+4>
cpsr    0x10    16
(gdb) stepi
5      MOV R2, #8
(gdb) info r
r0      0x1      1
r1      0x0      0
r2      0x0      0
r3      0x0      0
r4      0x0      0
r5      0x0      0
r6      0x0      0
r7      0x4      4
r8      0x0      0
r9      0x0      0
r10     0x0      0
r11     0x0      0
r12     0x0      0
sp      0x7efff690    0x7efff690
lr      0x0      0
pc      0x807c  0x807c <_start+8>
cpsr    0x10    16
(gdb)

```

Rimuoviamo il breakpoint con il comando `delete` e rifacciamo partire il programma con il comando `run`, dopo aver posto un breakpoint in `_start`.

Verifichiamo il valore dei registri con il comando `info r` anche dopo aver fatto un nuovo step.

## ATTIVITÀ 6

### Programmi ARM

Di seguito presentiamo il codice di alcuni esempi di programmi ARM.

#### 1 Strutture di controllo: selezione (if) e iterazione (while)

Vediamo ora il codice completo per l'implementazione della struttura di controllo `if`.

Consideriamo il seguente problema scritto con una pseudocodifica che generalizza un linguaggio ad alto livello qualsiasi:

```

if (a == b) then
    a = a + 1
else
    a = b
fine if

```

Utilizzando Assembly ARM possiamo implementare questo pseudocodice in modo tradizionale o con istruzioni condizionate che rendono il codice più semplice e compatto.

```

@implementa IF con metodo
tradizionale:
@if (a==b) then a=a+1 else
a=b fine if

.global _start
_start:

    mov r0, #10
    mov r1, #4

    cmp r0,r1
    beq then
    mov r0,r1
    b fine
then:
    add r0,r0,#1
fine:

_exit:
    mov r0,#0
    mov r7,#1
    swi 0

```

```

@implementa IF con istruzioni
condizionate:
@if (a==b) then a=a+1 else
a=b fine if

.global _start
_start:

    mov r0, #10
    mov r1, #4

    cmp r0,r1
    addeq r0,r0,#1
    movne r0,r1

_exit:
    mov r0,#0
    mov r7,#1
    swi 0

```

## 2 Strutture di controllo: iterazione (while)

Vediamo ora il codice completo per l'implementazione della struttura di controllo *while*. Consideriamo il seguente problema, che effettua lo scambio dei valori di *i* e *j*:

```

while ( i != 0 ) do
    i = i - 1
    j = j + 1
fine while

@ implementa while con metodo
tradizionale:
@ sia i -> r0, j -> r1
@ scambia i valori di r0 e r1

.global _start
_start:

    mov r0,#5
    mov r1,#0
while:
    cmp r0,#0
    beq fine
    sub r0,r0,#1
    add r1,r1,#1
    b while
fine:

_exit:
    mov r0,#0
    mov r7,#1
    swi 0

```

```

@ implementa while con
istruzioni condizionate:
@ sia i -> r0, j -> r1
@ scambia i valori di r0 e r1

.global _start
_start:

    mov r0,#5
    mov r1,#0
while:
    cmp r0,#0
    subne r0,r0,#1
    addne r1,r1,#1
    bne while

_exit:
    mov r0,#0
    mov r7,#1
    swi 0

```

**3 Iterazione: modifica del registro di stato**

Vediamo l'esecuzione di un ciclo, implementando due soluzioni: nella prima soluzione i bit del registro di stato sono modificati dall'istruzione `cmp`, nella seconda soluzione, più compatta, dall'istruzione `subs`.

**@esegue un ciclo 50 volte; i bit del registro di stato sono modificati da CMP**

```
.global _start
_start:
    mov r0, #1
loop:
    add r0,r0,#1
    cmp r0,#50
    ble loop

_exit:
    mov r0,#0
    mov r7,#1
    swi 0
```

**@esegue un ciclo 50 volte; i bit del registro di stato sono modificati da SUBS**

```
.global _start
_start:
    mov r0, #50
loop:
    subs r0,r0,#1
    bne loop

_exit:
    mov r0,#0
    mov r7,#1
    swi 0
```

**4 Selezione: calcolo del Massimo Comun Divisore (MCD)**

In questo esempio confrontiamo un algoritmo per il calcolo del MCD effettuato in modo tradizionale e un algoritmo per il calcolo del MCD che utilizza istruzioni condizionate. Nel secondo caso il codice è più semplice e compatto.

**@calcola MCD tra 10 e 4 con metodo tradizionale (MCD posto in r1)**

```
.global _start
_start:
    mov r0, #10
    mov r1, #4
MCD:
    cmp r0,r1
    beq FINE
    blt MIN
    sub r0,r0,r1
    b MCD
MIN:
    sub r1,r1,r0
    b MCD
FINE:

_exit:
    mov r0,#0
    mov r7,#1
    swi 0
```

**@calcola MCD tra 10 e 4 con istruzioni condizionate (MCD posto in r1)**

```
.global _start
_start:
    mov r0, #10
    mov r1, #4
MCD:
    cmp r0,r1
    subgt r0,r0,r1
    sublt r1,r1,r0
    bne MCD

_exit:
    mov r0,#0
    mov r7,#1
    swi 0
```

## 5 Vettori

Il seguente programma, utilizzando i vettori, scrive in *r1* la somma dei primi tre elementi del vettore di numeri interi, il cui indirizzo base è contenuto in *r0*.

**@ scriviamo in r1 la somma dei primi 3 elementi del vettore il cui indirizzo base è contenuto in r0**

```
.global _start
_start:
.data
numeri: .word 2,4,6,8

.text
main:
ldr r0,=numeri    @carico in r0 l'indirizzo di inizio del vettore
                   numeri
ldr r1,[r0]        @carico in r1 il numero 2
ldr r2,[r0,#4]     @carico in r2 il numero 4
add r1,r1,r2      @sommo il contenuto di r1 e r2 e metto
                   il risultato in r1 (r1=6)
ldr r2,[r0,#8]     @carico in r2 il valore 6
add r1,r1,r2      @sommo il contenuto di r1 e r2 e metto
                   il risultato in r1 (r1=8)

_exit:
mov r0,#0
mov r7,#1
swi 0
.end
```

## 6 Lettura da memoria: vettori di caratteri e di numeri interi

Vediamo ora esempi di lettura dalla memoria.

```
.global _start
_start:
.data
pari: .word 2,4,6,8
string: .asciz "i primi numeri pari"

.text
main:
ldr r0,=pari    @carico in r0 l'indirizzo di inizio del vettore
                 pari
ldr r1,[r0]      @carico in r1 il numero 2
ldr r2,[r0,#4]   @carico in r2 il numero 4
ldr r0,=string   @carico in r0 l'indirizzo di inizio della stringa
                 string
ldrb r3,[r0]      @carico in r3 la lettera i (codice ASCII 105)
ldrb r4,[r0,#3]   @carico in r4 la lettera p

_exit:
mov r0,#0
mov r7,#1
swi 0
.end
```

## 7 Stampa di una stringa

Per stampare una stringa a video, occorre:

- caricare in r7 il servizio 4;
- caricare in r1 l'indirizzo di inizio della stringa da stampare;
- caricare in r2 la lunghezza della stringa da stampare.

Il seguente programma stampa a video la stringa «stringa da stampare»:

- r0 identifica la periferica di output, in questo caso 1, che è il monitor;
- r1 contiene l'indirizzo di memoria in cui è posto il primo carattere della stringa da stampare;
- r2 contiene il numero di caratteri da visualizzare, in questo caso 20 (compresi gli spazi e il carattere «\n» per andare a capo);
- r7 contiene il numero della funzione da eseguire (in questo caso 4);
- per allocare la stringa è possibile utilizzare .asciz e non mettere «\n» nella stringa, che è comunque contato come carattere della stringa (codice ASCII 10).

```
global _start
_start:
    mov r7,#4
    mov r0,#1
    ldr r1,=string
    mov r2,#20
    swi 0

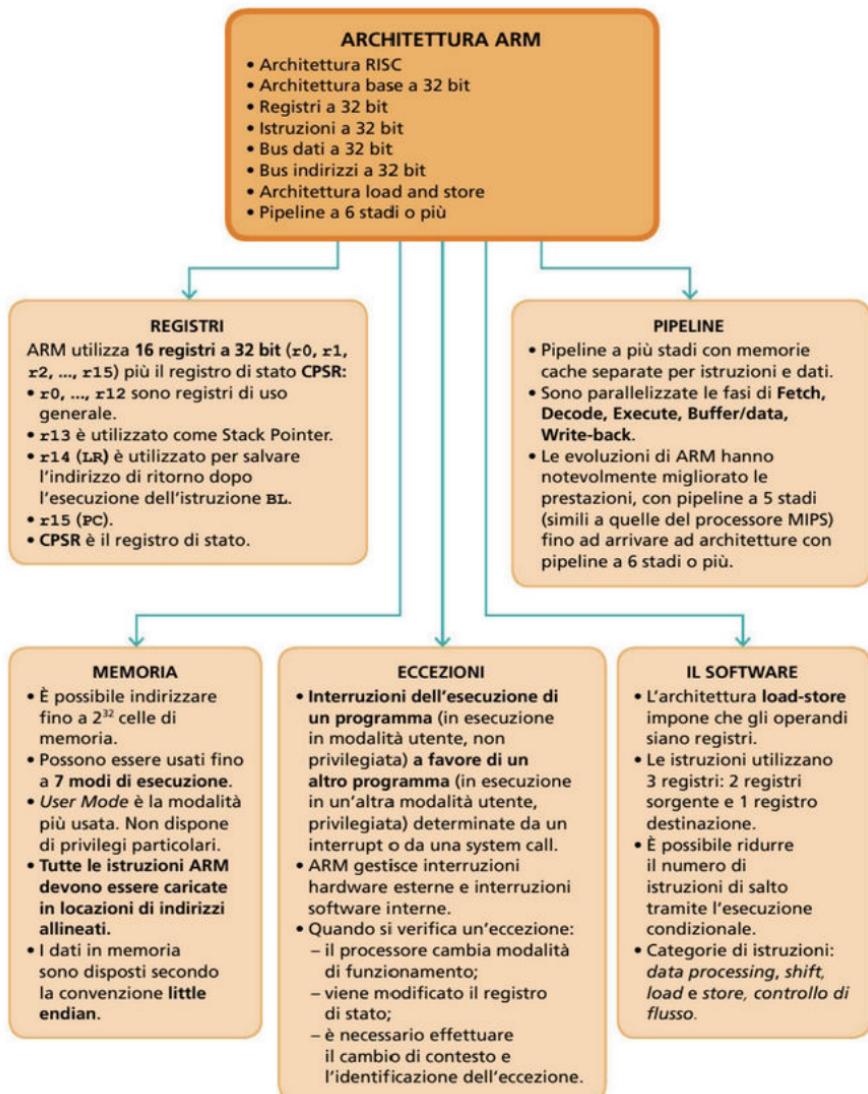
    _exit:
    mov r7,#1
    swi 0

.data
string: .ascii "stringa da stampare\n"
```



**File aggiuntivi**  
Codice completo  
pronto per  
l'esecuzione

# SINTESI



# Verifica delle CONOSCENZE

Unità 8



## Vero o Falso?

- 1 ARM è un'architettura RISC.
- 2 Il registro CPSR di ARM è lo stack pointer.
- 3 In ARM gli operandi di un'istruzione possono essere tutti indirizzi di memoria.
- 4 La memoria ARM è organizzata in parole di 32 bit.
- 5 Il registro r15 è usato come Program Counter.
- 6 ARM non ha Program Counter perché le istruzioni hanno sempre la stessa dimensione.
- 7 Le istruzioni ARM devono essere caricate a indirizzi allineati.
- 8 L'organizzazione della memoria del processore ARM è little endian.
- 9 Le istruzioni ARM hanno sempre due operandi.
- 10 Un'istruzione può coinvolgere il trasferimento di registri multipli.



## Rispondi ai seguenti quesiti indicando l'unica risposta esatta.

- 11 ARM è dotato di
- a. 32 registri di 16 bit
  - b. 15 registri di 32 bit
  - c. 16 registri di 32 bit
  - d. 32 registri di 8 bit
- Domande per la prova orale
- 18 Dove viene utilizzato il processore ARM e perché?
- 19 Quando e come vengono modificati i bit del registro di stato?
- 20 Qual è il formato delle istruzioni di data processing che modificano il registro di stato?
- 21 Cosa significa «istruzione condizionata»?
- 22 Quali sono le categorie di istruzioni Assembly ARM?
- 12 Le istruzioni ARM di data processing sono:
- a. aritmetiche, logiche, di trasferimento
  - b. di trasferimento
  - c. di shift
  - d. di salto
- 13 Il registro r14
- a. è usato come registro di stato
  - b. può essere usato come stack pointer
  - c. è usato come program counter
  - d. è usato per salvare l'indirizzo di ritorno da un sottoprogramma
- 14 Si consideri l'istruzione add r0,r1,r2:
- a. non è corretta perché ha tre operandi
  - b. r0 è il registro destinazione
  - c. r2 è il registro destinazione
  - d. r2 contiene il riporto della somma
- 15 Si consideri l'istruzione ldr r0,[r1]:
- a. trasferisce in r1 i 4 byte di memoria a partire dall'indirizzo contenuto in r0
  - b. trasferisce in r1 1 byte di memoria posto dall'indirizzo contenuto in r0
  - c. trasferisce in r0 i 4 byte di memoria a partire dall'indirizzo contenuto in r1
  - d. trasferisce in r0 1 byte di memoria posto dall'indirizzo contenuto in r1
- 16 L'istruzione moveq r0,#1:
- a. è un'istruzione di salto
  - b. è un'istruzione aritmetica
  - c. è un'istruzione condizionale
  - d. è un'istruzione di shift
- 17 Le eccezioni:
- a. sono degli interrupt
  - b. possono essere solo software
  - c. aumentano le prestazioni del sistema
  - d. implementano le strutture di controllo
- 23 Quali sono le caratteristiche di ARM tipiche di un'architettura RISC?
- 24 Cosa significa «trasferimento di registri multipli»?
- 25 Cosa significa «architettura load/store»?
- 26 Cosa sono e come sono le gestite le eccezioni in ARM?

- 27** Scrivi in linguaggio Assembly ARM le seguenti istruzioni espresse in linguaggio naturale.
- Trasferire nel registro  $r3$  il contenuto della locazione di memoria di indirizzo EC34h
  - Trasferire nella locazione di memoria di indirizzo 0026h il valore 68
  - Sommare al contenuto del registro  $r3$  il valore 64h e mettere il risultato in  $r3$
  - Incrementare il valore del registro  $r3$
  - Sommare al contenuto della locazione di memoria di indirizzo 0F56h il valore 0002h
- 28** Descrivi il significato dell'istruzione  
stm  $r0,[r2, r4-r7]$ .
- 29** Descrivi il significato del seguente programma e rispondi alle domande:
- ```
mov r0, #8
mov r1, #2
cmp r0,r1
adreq r0,r0,#1
movne r0,r1
```
- cosa contiene il registro  $r0$  al termine del programma?
  - Cosa contiene il registro  $r1$  al termine del programma?
  - Quali bit del registro di stato sono coinvolti?
  - Quali istruzioni utilizzano i bit del registro di stato?
- 30** Per ogni categoria di istruzioni ARM, fai un esempio di istruzione.
- 31** Scrivi un programma ARM per sommare i valori corrispondenti di due vettori di 4 byte.
- 32** Scrivi un programma in Assembly ARM che stampa a video la stringa «ciao mondo!»
- 33** Descrivi il significato dei 4 bit più significativi di un'istruzione ARM facendo anche un esempio nel caso in cui valgano 1110.
- 34** Scrivi un'istruzione utilizzando un software. Verifica la sua codifica esadecimale. Traducila in binario e spiega il significato dei 32 bit.
- 35** Spiega la differenza tra le istruzioni add, adds, addgt.
- 36** Spiega la differenza tra le istruzioni mov, ldr, str.
- 37** Perché per le operazioni aritmetiche basterebbe una pipeline a 3 stadi, mentre per le load/store sarebbe opportuno avere un maggior numero di stadi?
- 38** Scrivi delle istruzioni ARM che modifichino i flag NZC del registro di stato.





## ARM Architecture

### Abstract

ARM is a RISC instruction set architecture. The base architecture operates on 32 bit. The instructions, data-bus, address bus, and registers, all operate on 32 bit. The address bus allows the processor to address up to 232 memory cells. Bytes in memory are stored in little endian order and instructions

are loaded into memory aligned.

The instruction set includes instructions for data processing, shift, load and store, and jump. Code can be optimised with conditional execution. ARM architecture is very popular in mobile devices and Raspberry Pi.

### Questions

#### Answer all questions

- 1 Describe a generic ARM CPU instruction.
- 2 What are the shift instructions?
- 3 What are the registers in ARM?
- 4 How is memory handled in an ARM CPU?

#### Choose the correct answer



- 5 ARM:
- [a] uses 32-bit address bus
  - [b] uses 16-bit data bus
  - [c] doesn't support direct addressing
  - [d] doesn't support interrupt

- 6 ARM is a RISC architecture, and therefore

- [a] has a pipeline
- [b] manages exceptions
- [c] all instructions have the same size
- [d] includes instructions to address multiple registers

- 7 The swi instruction

- [a] generates a software interrupt
- [b] generates a hardware interrupt
- [c] writes a string to terminal
- [d] reads a string from memory

- 8 The bne instruction

- [a] is an exception
- [b] is a conditional instruction
- [c] is a jump instruction
- [d] is a logic instruction

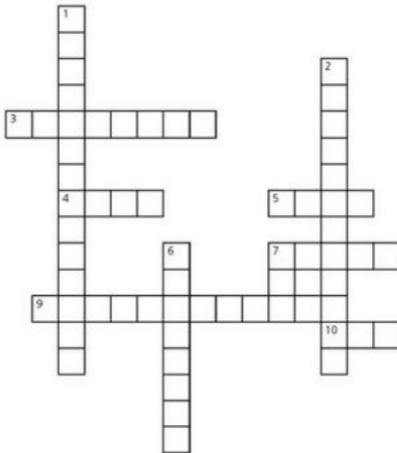
### Crosswords

#### Across

- 3 Technique to speed up the execution of a programme.
- 4 Status register.
- 5 A CPU-internal interrupt.
- 7 A category of instructions that move the operand by 1 bit.
- 9 Representation mode of data in memory.
- 10 An instruction used for constants.

#### Down

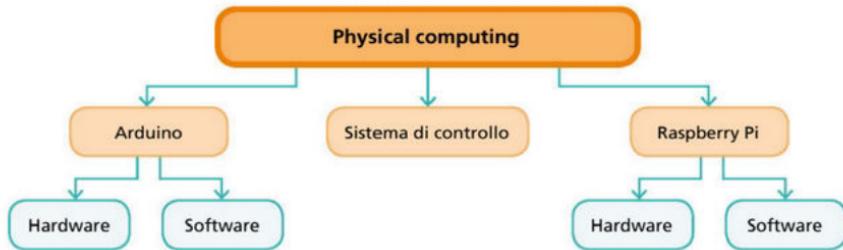
- 1 A family of instructions that include arithmetic instructions.
- 2 How instructions are loaded into memory.
- 6 An execution mode with no privileges.
- 7 This instruction generates a software interrupt.



# Unità 9



## Il Physical Computing



### Visione d'insieme

- Sistemi di controllo per l'interazione con il mondo fisico che ci circonda.
- Primi passi nell'ambito dell'Internet of Things (IoT).
- I microcontrollori Arduino e Raspberry Pi.



Fig. 1. L'ecosistema Internet of Things.

## 1 Introduzione

Il **Physical Computing** è costituito da sistemi hardware e software in grado di **percepire**, grazie alla rilevazione di parametri fisici tramite sensori, il mondo che li circonda e di **interagire** con esso.

La rilevazione dei parametri fisici tramite sensori non è una novità, ma in questi ultimi anni, grazie alla convergenza delle nuove tecnologie con Internet, ha ripreso vigore e sta cambiando lo scenario mondiale. L'introduzione di progetti open source e delle schede a microcontrollore a basso consumo (come Raspberry Pi e Arduino) ha accelerato il processo di cambiamento.

L'utilizzo dei dispositivi è passato da un contesto prevalentemente locale a una rete di oggetti che interagiscono dando vita a un ecosistema, chiamato **Internet delle cose (IoT, Internet of Things)** di cui fanno parte anche gli esseri umani (Figura 1); tutti gli elementi dell'ecosistema sono connessi tramite reti di comunicazione.

## 2 Introduzione all'Internet of Things (IoT)

### Pit Stop

- Che cos'è il Physical Computing?
- Che cos'è l'IoT?
- In quali modi possono essere elaborati i dati?

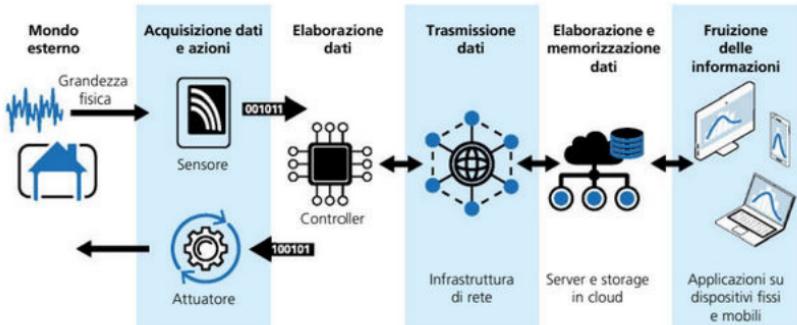
Fig. 2 La filiera dei dati: sensore-microcontrollore-rete-server-utente finale. Il sensore è l'elemento primario della catena che si conclude con l'arrivo dei dati a un server in cloud, che estrae le informazioni utili.

Le «cose» dell'IoT sono gli **oggetti intelligenti (smart objects)**, dispositivi di varia natura in grado di collegarsi alla rete Internet e di interagire con il mondo esterno.

Lo scenario è quello di un ecosistema costituito da comunità di oggetti e persone che interagiscono tra loro e con l'ambiente che li circonda. Elemento centrale di questo complesso sistema sono i **dati**. I vari elementi dell'IoT possono essere molto diversi tra loro, ma tutti hanno a che fare con i dati.

In generale, una catena completa parte dalla raccolta dei dati. Questi possono essere elaborati localmente, al fine di agire sul mondo esterno nell'immediato, oppure possono essere trasmessi a dispositivi remoti in grado di memorizzarli, elaborarli, analizzarli, al fine di ottenere informazioni utili (Figura 2).

L'IoT è uno dei modi per generare dati. Altre fonti sono, per esempio, i social media, le transazioni aziendali, che usano database (per esempio lo scontrino della spesa, l'acquisto di un biglietto aereo, il registro elettronico), la navigazione nel web, gli esperimenti scientifici. Tutti questi dati sono oggetto di una importante disciplina: la **scienza dei dati**.



La possibilità di trarre informazioni e conoscenza dai dati sta avendo un forte impatto sulla nostra vita quotidiana. Siamo in grado di intervenire meglio sull'ambiente, di risparmiare energia, di prenderci cura in modi nuovi delle persone, di facilitare il traffico, di rendere più efficienti i parcheggi, di rendere più

confortevoli le abitazioni. Il punto centrale rimane il limite, sempre incerto, tra la possibilità di intervenire in modo positivo su vari aspetti della nostra esistenza e il rischio di perdere la nostra identità e la nostra libertà.

In questa unità studieremo come funziona la parte iniziale della catena dei dati: il passaggio dal mondo fisico in cui viviamo alla sua trasformazione in dati tramite i **sensori** e le **azioni di controllo**.

### 3 Il controllo delle grandezze fisiche

Elementi base del Physical Computing sono i **sensori**, gli **attuatori** e i **controllori**.

Le grandezze fisiche, cioè gli elementi misurabili che caratterizzano i fenomeni fisici, come per esempio la temperatura o la pressione, vengono rilevate tramite sensori, convertite in segnali digitali e inviate al controllore (microcontrollore) per la loro elaborazione. I risultati ottenuti dall'elaborazione possono essere usati per pilotare, tramite attuatori, dispositivi esterni come, per esempio, sistemi elettromeccanici e motori (Figura 3).



#### Pit Stop

- 4 Descrivi la funzione di ognuno dei tre elementi base del Physical Computing.

◀ Fig. 3 Le grandezze fisiche sono rilevate da sensori, convertite in segnali elettrici e inviate al sistema di controllo per la loro elaborazione. Il controllore ordina agli attuatori il pilotaggio di dispositivi esterni.

#### Sensori

Un sensore è un dispositivo che:

- **rileva e misura** le proprietà fisiche dell'ambiente (per esempio movimento, posizione, distanza, luminosità, suono, temperatura, umidità, pressione);
- **trasforma** le proprietà fisiche in un segnale di natura elettrica;
- **fornisce** il segnale di natura elettrica al sistema di elaborazione.

Esempi di sensori sono:

- il microfono, che converte l'energia dell'onda sonora in un segnale elettrico;
- una termoresistenza, che sfrutta la resistività di un conduttore, che a sua volta – variando al variare della temperatura – consente di risalire alla temperatura del corpo;
- i molti sensori contenuti in un cellulare: per esempio, l'accelerometro (un sensore che misura l'accelerazione), il giroscopio (un sensore che serve a misurare l'orientamento spaziale), la bussola (un sensore che misura la forza e la direzione di un campo magnetico), il GPS (un sensore che fornisce a un ricevitore informazioni sulle sue coordinate geografiche), il sensore di luminosità.

#### Pit Stop

- 5 Quali sono le funzioni di un sensore?

**Pit Stop**

- 6 Perché i segnali continui devono essere convertiti in segnali discreti?

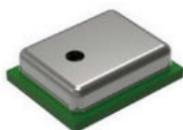


Fig. 4 Microfono MEMS per cellulare.

Spesso le grandezze fisiche assumono valori continui nel tempo. Dato che i calcolatori trattano solo valori discreti, i segnali analogici (continui) devono essere campionati (cioè si registrano i valori solo in corrispondenza di istanti di tempo definiti) e convertiti in segnali digitali per poter essere manipolabili dal computer. La conversione può essere realizzata tramite appositi circuiti oppure dai sensori stessi, come capita per esempio per i microfoni, che presentano un'interfaccia digitale.

Con l'avvento dei **MEMS** (*Micro Electro Mechanical Systems*) i sensori sono stati inseriti direttamente nei chip di silicio e sono diventati onnipresenti. Per esempio, la **Figura 4** mostra un microfono per cellulare in cui il diaframma è sensibile alla pressione della voce. Il segnale rilevato viene amplificato e convertito in digitale.

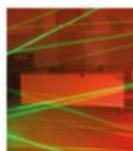
Di seguito sono elencati alcuni esempi di sensori di uso comune.

**Sensore di luminosità** (*light sensor*) Fotocellule, fotodiodi, fototransistor, *encoder* ottici.

Rileva l'intensità della luce e la lunghezza d'onda.



Luminosità degli ambienti esterni e interni: la luce dei lampi si accende quando la luminosità ambientale assume un valore inferiore a un valore prefissato.



Sistemi di sicurezza: un sensore rileva l'interruzione di un fascio luminoso, il controllore segnala la violazione del sistema attivando un allarme.



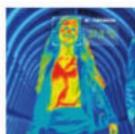
Scanner di *bar code* o *QR code*: nelle parti bianche la luce viene riflessa più che in quelle scure; il dispositivo converte la luce riflessa in un codice binario che può essere letto dal computer.



Scanner retinici: sulla retina è effettuata una scansione tramite raggi infrarossi per acquisire la mappa dei vasi sanguigni. Sono usati in ambito medico o per i dispositivi di sicurezza biometrici digitali.

**Sensore a infrarossi (infrared sensor)**

Rileva i raggi infrarossi irradiati da persone o oggetti.



Rilevando la temperatura di un oggetto tramite le radiazioni infrarosse, applicato a un «convertitore» può mostrare colori visibili all'occhio umano.



In astronomia serve a fotografare oggetti troppo grandi per emettere luce visibile.

**Sensore di temperatura e calore**

Rileva la temperatura di un oggetto (termometri, termocoppie, termostati, calorimetri).



È utilizzato nei termostati, che, per contatto diretto, misurano il valore della temperatura dell'ambiente.



È utilizzato per la misurazione della temperatura del gas di scarico delle auto, correlata al livello di emissioni inquinanti.

**Sensore di movimento (motion sensor)**

Può utilizzare una tecnologia a infrarossi (rileva il calore di un corpo) oppure le microonde (la ricezione di un fascio di microonde viene interrotta dal passaggio di un corpo in movimento).



Porte automatiche: quando una persona o un animale si avvicina alla porta, questa si apre.



Sistemi di sicurezza dei musei: per rilevare un intruso e per accendere luci quando viene rilevata una presenza estranea.

**Sensore di umidità** (*moisture sensor, humidity sensor*)

Rileva il grado di umidità. *Moisture sensor*: rileva il grado di umidità nel terreno. *Humidity sensor*: rileva il grado di umidità nell'aria.



Utilizzato per misurare l'umidità di un ambiente: può controllare che l'umidità dell'ambiente non superi un determinato valore di soglia, al di sopra del quale potrebbe nuocere alla salute oppure danneggiare organismi o oggetti quali opere d'arte.



È utilizzato in agricoltura per misurare il livello di umidità del terreno, così da rendere il terreno adeguato al tipo di pianta coltivata. Permette, per esempio, di creare un sistema automatizzato di irrigazione dei campi.

**Sensore di pH (acidità)**

Misura quanto è acida o alcalina una sostanza. Utilizza come riferimento una scala numerica: se il valore è 7, la sostanza ha un pH neutro; se il valore è inferiore a 7, la sostanza ha un pH acido; se invece è superiore a 7, la sostanza ha un pH alcalino (o basico).



Controllo del livello del pH nell'acqua che beviamo (che deve avere un pH neutro), e negli acquari, affinché il pH sia costante e simile a quello dell'habitat naturale dei pesci che contengono.

**Sensore di gas e fumo**

Rileva la presenza di gas nell'ambiente, controllando che non attraversi determinati valori di soglia.



Rilevazione di eventuali fughe di gas: rileva il monossido di carbonio, o altri tipi di gas. Usato anche per controllare che l'acqua utilizzata per innaffiare le piante sia priva di gas nocivi.

**Sensore di prossimità**

Realizzati con sensori: induttivi, capacitivi, magnetici, a ultrasuoni, a infrarossi.

Rileva la presenza di corpi o oggetti nelle immediate vicinanze. Spesso è installato su macchinari industriali.



Sensori a ultrasuoni: il trasmettitore a ultrasuoni emette un breve impulso sonoro in una particolare direzione. L'impulso rimbalza su un bersaglio e ritorna al ricevitore dopo un intervallo di tempo. Usati per il rilevamento di oggetti, la misurazione del livello di un liquido in un contenitore, il rilevamento di anomalie.



Parcheggio: il sensore a infrarossi posto sul soffitto di un parcheggio rileva l'occupazione della piazzola.



Dispenser igienizzante per le mani a infrarossi: rileva la differenza di temperatura tra la mano e l'ambiente.



Sensore wireless automatico per auto intelligente: rileva la distanza degli ostacoli.



Segnalatori di presenza: usati per ottimizzare gli impianti di illuminazione, evitare sprechi di energia e ridurre i costi o per rilevare presenze indesiderate.



I sensori di prossimità sono utilizzati spesso su linee di produzione industriale per il conteggio dei pezzi, il riconoscimento dei codici e il controllo della sicurezza.

## Esempio 1

### Il termometro: un sensore che tutti abbiamo imparato a conoscere

Nella prima infanzia mi capitava spesso di ammalarmi. Il termometro a mercurio che veniva conservato in un cassetto era diventato l'oggetto che avevo imparato a temere di più, ma anche a conoscere meglio. La misura della temperatura corporea era rilevata da quello che, più tardi, avrei riconosciuto essere un **sensore**. La dilatazione del mercurio contenuto nell'ampolla del termometro è proporzionale alla variazione di temperatura; possiamo leggere la temperatura confrontando la lunghezza della colonna di mercurio con le tacche sulla scala graduata (**Figura 5a**). La precisione della misura è limitata, ma sufficiente per il nostro scopo, anche se non è particolarmente utile per le applicazioni di acquisizione di dati in cui i valori di uscita devono essere digitalizzati. Per questo dovremmo ricorrere ai termometri digitali (**Figura 5b**) nei quali il valore della temperatura è convertito in valori discreti che, in linea teorica, potremmo archiviare e analizzare.



↑ Fig. 5a Esempio di un termometro analogico a mercurio. Un termometro analogico a mercurio è costituito da un tubo di vetro con una scala graduata, che contiene una colonna di mercurio. b Esempio di un termometro digitale. Un termometro digitale è dotato di un sensore che trasmette la temperatura a un microcircuito, che la visualizza su un piccolo display a cristalli liquidi.

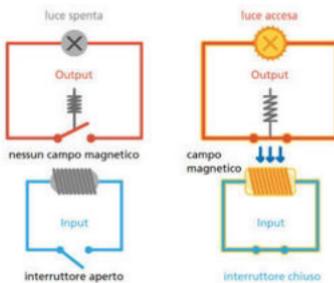
## ■ Attuatori

Un **attuatore** agisce sul mondo esterno e lo modifica, trasformando segnali elettrici in grandezze fisiche. Gli attuatori sono comandati dal **controllore** che permette loro di eseguire azioni come, per esempio, aprire una valvola, azionare un motore, fornire informazioni all'utente, ecc.

Di seguito sono elencati alcuni esempi di attuatori.

### Relè (attuatore elettromagnetico)

Un relè è costituito da un solenoide che, quando è attraversato da una corrente, genera un campo magnetico che attrae un'armatura metallica: l'armatura muovendosi chiude o apre un interruttore.



Modulo relè per la scheda elettronica Arduino.

### Servomotore (attuatore elettromeccanico)

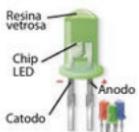
Un servomotore è un particolare motore elettrico che controlla con precisione la rotazione del perno del motore per fargli assumere un preciso angolo, oltre che controllare la velocità e l'accelerazione. È un attuatore elettromeccanico perché converte l'energia elettrica in energia meccanica e viceversa. Il campo elettromagnetico permette di accoppiare il sistema elettrico con quello meccanico.



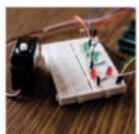
Il servomotore è spesso utilizzato per la regolazione della posizione nei sistemi di robotica. La figura mostra un servomotore con ingranaggi in metallo per progetti robotici.

### LED - Light Emitting Diode (Attuatori opto-elettronici)

Il LED a semiconduttore converte segnali elettrici in segnali luminosi.



Il LED a semiconduttore sfrutta un fenomeno quantistico chiamato «ricombinazione elettrone-lacuna». Applicando al LED una differenza di potenziale, la ricombinazione induce gli elettroni a emettere fotoni di energia ben definita, corrispondente a un preciso colore emesso dal LED.



Breadboard con LED, fotoresistenze e servomotore connessi ad Arduino. I LED sono molto utilizzati come indicatori luminosi nelle schede di controllo, negli elettrodomestici e macchinari, ma anche per l'illuminazione di grandi spazi. Hanno consumi molto bassi, perché non generano calore e la perdita di potenza per riscaldamento è praticamente nulla.

### Altoparlanti MEMS (Attuatori acustici)

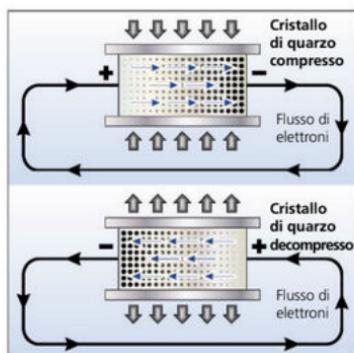
Un altoparlante MEMS converte segnali elettrici in suono generando onde sonore udibili dall'orecchio umano. La membrana che produce le onde sonore è deviata da un attuatore piezoelettrico.



Microaltoparlante MEMS per cuffie wireless. Gli altoparlanti MEMS sono presenti in applicazioni audio dove le piccole dimensioni sono fondamentali: smartphone, auricolari wireless, cuffie, dispositivi indossabili.

### Cristallo piezoelettrico (Attuatore piezoelettrico)

Un cristallo piezoelettrico si deforma se viene sottoposto a un campo elettrico e viceversa. La deformazione meccanica del cristallo provoca una differenza di potenziale e il passaggio di una corrente elettrica.



Effetto piezoelettrico di un cristallo con la generazione di una corrente elettrica ( $i$ ).



Cicalino piezoelettrico. I cicalini piezoelettrici sono presenti nei trasduttori elettroacustici (altoparlanti e microfoni) ed elettromeccanici.

## ■ Controllori

Sensori e attuatori costituiscono il collegamento con il mondo fisico, ma senza un'«intelligenza» che li guida non sono utili.

Per esempio, per controllare la traiettoria e la velocità di un autoveicolo (Figura 6a), non basta impostare il luogo di destinazione e l'ora di arrivo, ma occorre intervenire continuamente sulla posizione del volante, dell'acceleratore e degli elementi del sistema frenante. Infatti, anche nel caso di perfetta efficienza del mezzo e di strada rettilinea, non saremmo in grado di prevedere gli effetti di disturbi esterni quali vento, pioggia, asperità del terreno e temperatura dell'asfalto, condizioni che andrebbero a perturbare il nostro sistema e sarebbero fonti di errori.

Controllare un sistema significa verificare che il comportamento del sistema (per esempio, un impianto o un generico dispositivo) sia conforme alle aspettative e, in caso contrario, intervenire con azioni di correzione che riportano il sistema alle condizioni desiderate (Figura 6b).

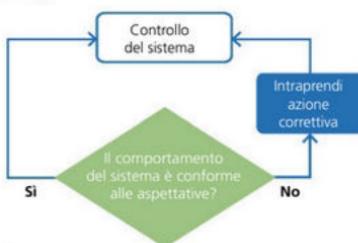
Fig. 6a Le azioni che vengono effettuate sull'autoveicolo (accelerazione, frenata e guida) fanno in modo che la velocità e la direzione mantengano i valori desiderati.

b L'algoritmo di controllo di un sistema: l'algoritmo compie la verifica ciclica del comportamento del sistema e la conseguente azione di correzione.



- IN**
- direzione
  - ostacoli
  - velocità
  - precedenze
- OUT**
- direzione corretta
  - ostacolo evitato
  - velocità controllata
  - precedenza corretta

a.



b.

Il **controllore** (o controller) è il dispositivo che guida il sistema a raggiungere il suo scopo.

Ci sono due tipi di controllo:

- il controllo **ad anello chiuso**;
- il controllo **ad anello aperto**.

### Controllo ad anello chiuso

Il controllo ad anello chiuso (*closed loop*) agisce tramite un meccanismo di retroazione (*feedback*) che permette al sistema di autoregolarsi:

- la **retroazione negativa** (*negative feedback*) avviene quando in un sistema da controllare il segnale di uscita è riportato al controllore e confrontato con l'input di riferimento (*setpoint*). La differenza tra il valore di riferimento in ingresso e il segnale misurato dal sensore genera un «errore», che viene utilizzato dal controllore per comandare gli attuatori e per riportare il sistema alle condizioni desiderate (Figura 7a);
- la **retroazione positiva** (*positive feedback, self-reinforcing feedback*) avviene quando il segnale di uscita è sommato parzialmente al segnale di ingresso per rinforzarlo (Figura 7b).

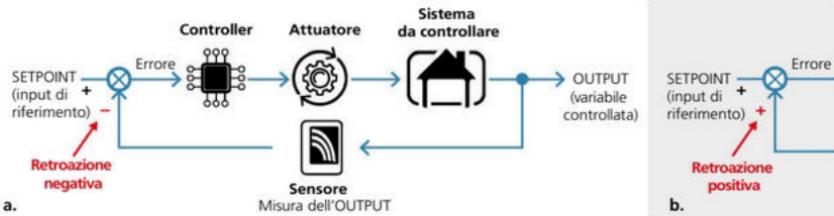


Fig. 7a Controllo ad anello chiuso con retroazione negativa. Si effettua la somma algebrica tra i due valori. b Controllo ad anello chiuso con retroazione positiva. Si effettua la somma tra i due valori.

### Comprendi con l'analogia

Controllare un sistema di controllo con retroazione negativa è come salire le scale controllando dove si mettono i piedi

Ho osservato spesso il mio compagno Gabriele salire al mattino le rampe di scale che lo portano alla sua classe al terzo piano, la Terza B. I suoi passi sono lenti e pesanti. Prima di appoggiare un piede sembra pensarsi due volte: osserva con attenzione il gradino, controllo il tipo di materiale e valuta se potrà sostenere il suo peso. Solo dopo qualche istante lascia cadere il piede, sconsolato.

Ha calcolato che ci mette più di tre minuti dall'entrata della scuola a quando varca la soglia della classe.

Quando suona la campanella dell'ultima ora le cose vanno molto diversamente. Gabriele esce dall'aula come una furia e si precipita per le scale. Non vede neppure i gradini e, ne sono certo, se anche ne mancasse qualcuno neppure se ne accorgerebbe, tanta è la premura di uscire. Il tempo medio di discesa è inferiore al minuto, nonostante l'intralcio costituito dalla fiumana di studenti che si accalcano all'uscita (le porte delle scuole sono sempre troppo grandi per entrare e troppo piccole per uscire!).

Salire le scale con attenzione e controllare dove si posa il piede, cioè rilevare con un sensore (l'occhio) lo stato del sistema per informare il controller (il cervello) è un ottimo esempio di retroazione negativa.

Al contrario, agire senza alcun controllo è un esempio di anello aperto. Questo è anche ciò che succede quando si tira un calcio di rigore: quando il pallone è calcio non c'è verso di modificarne la traiettoria. Non basta mirare con precisione. È sufficiente un alito di vento o una leggera increspatura del terreno che ci fa scivolare e non siamo più in grado di prevedere dove finirà il pallone.



Il meccanismo del feedback negativo è quello che ci interessa maggiormente per i risvolti che ha nei sistemi automatici di controllo. L'esempio relativo al controllo della temperatura della doccia (che mostriamo sotto) descrive un sistema ad anello chiuso in cui la regolazione è manuale: una persona valuta la temperatura dell'acqua e si comporta di conseguenza.

Processi di controllo di questo tipo sono replicabili in modo automatico dagli elaboratori.

Gli algoritmi di controllo possono essere molto sofisticati e in grado di rilevare gli errori di scostamento dagli obiettivi e intervenire tempestivamente per correggerli, come capita nei regolatori **PID** (**P**roporzionale **I**ntegrale **D**erivativo) utilizzati, per esempio, nella movimentazione di un braccio robotico o nel posizionamento di un drone.

Nei regolatori PID (si veda l'Approfondimento «Il controllo PID») l'uscita è generata valutando il contributo di tre termini (Figura 8):

- **proporzionale:** il segnale dell'uscita del controllore è proporzionale all'errore;
- **integrale:** il segnale dell'uscita del controllore è proporzionale all'accumulo degli errori in un certo periodo di tempo, cioè all'integrale dell'errore. I risultati degli errori vengono continuamente misurati e sommati. Quanto più a lungo persiste l'errore, tanto più la somma cresce e può essere utilizzata come controllo;
- **derivativo:** il segnale dell'uscita del controllore è proporzionale alla velocità di cambiamento dell'errore, cioè alla derivata dell'errore.

### Pit Stop

**10** Descrivi il significato dell'acronimo PID.

**11** Quali sono i termini che vengono valutati da un regolatore PID?

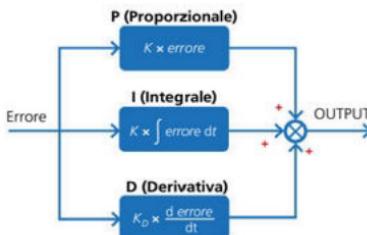


Fig. 8 L'uscita è la somma di tre contributi dell'errore: proporzionale, integrale e derivativo.

### Esempio 2

#### Retroazione negativa: il controllo della temperatura della doccia

Mentre siamo sotto la doccia i sensori della pelle rilevano la temperatura dell'acqua. L'informazione passa al cervello, che comanda alla mano di azionare il miscelatore. La persona sotto la doccia chiude l'anello di feedback con una **retroazione negativa** (Figura 9).



Fig. 9 Feedback negativo. La regolazione della temperatura della doccia.

Quando abbiamo fretta e cerchiamo di ottenere una temperatura confortevole dell'acqua, cominciamo a girare in modo compulsivo il miscelatore portandolo al massimo e, appena l'acqua diventa troppo calda, lo riportiamo al minimo. Questo sistema peggiora le cose e fa oscillare il sistema intorno alla temperatura desiderata senza raggiungerla mai (**Figura 10a**).

Che cosa dobbiamo fare per controllare il sistema e portare l'acqua alla temperatura desiderata in un tempo ragionevole? L'algoritmo corretto che applichiamo intuitivamente ha lo scopo di minimizzare l'errore, cioè di fare in modo che la differenza tra l'uscita effettiva e quella desiderata sia la più piccola possibile, in modo tale che il sistema raggiunga l'equilibrio (**Figura 10b**):

- In primo luogo **impostiamo la temperatura** dell'acqua ruotando la manopola del miscelatore nella posizione corrispondente alla temperatura che riteniamo ideale.
- Poi mettiamo la mano sotto il getto dell'acqua e **confrontiamo la sua temperatura con quella che ci aspettiamo**.
- **Se** la temperatura è quella desiderata, non facciamo nulla, se non tornare periodicamente a controllare la temperatura.
- **Altrimenti** agiamo delicatamente sulla leva del miscelatore per correggere l'errore:
  - **se** l'acqua è troppo calda, ruotiamo dalla parte del «freddo»;
  - **se** l'acqua è troppo fredda, ruotiamo dalla parte del «caldo».



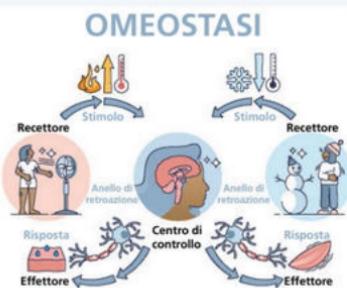
† Fig. 10a Un controllo non adeguato può far oscillare il sistema, che non raggiunge mai l'equilibrio. b La retroazione negativa tende a stabilizzare il sistema, raggiungendo un livello di equilibrio. In questo caso la temperatura, che è impostata su 35 °C, viene raggiunta dopo un determinato tempo.

### Esempio 3

**L'omeostasi, il meccanismo di controllo del nostro corpo, è un sistema a retroazione negativa**

Le sensazioni che proviamo (siano esse olfattive, visive, uditive o dolorose) dipendono da speciali cellule, i recettori, che appartengono al sistema nervoso periferico.

I recettori sono in grado di convertire gli stimoli provenienti dall'ambiente esterno in segnali elettrici inviati al cervello. Per esempio i recettori tattili della cute convertono l'energia meccanica, mentre i fotorecettori della retina convertono l'energia luminosa. Come avviene nei sensori, questo processo di trasduzione converte tutte le forme di energia nello stesso tipo di segnale elettrico, che viene inviato alle diverse aree cerebrali specializzate per elaborarlo.



† Fig. 11 Meccanismo di omeostasi nel corpo umano.

Una delle caratteristiche più importanti che hanno gli esseri viventi è la capacità di autoregolarsi per mantenere costanti i parametri vitali interni nonostante le variazioni dell'ambiente esterno (omeostasi). Ne è un esempio la temperatura corporea, che nella specie umana si mantiene tra i 36 e i 37 °C (Figura 11). Il cervello-controllore interpreta lo stimolo che arriva dai recettori nervosi della pelle (sensori) e reagisce prontamente con una risposta che serve a mantenere l'equilibrio interno: se fa troppo caldo il corpo suda e la temperatura si abbassa, se fa freddo il corpo è scosso da brividi per ottenere un movimento che produce un po' di calore. Il meccanismo si basa sulla retroazione negativa, in cui lo stimolo che genera la risposta viene annullato dalla risposta stessa.

### Controllo ad anello aperto

Il controllo ad **anello aperto** non ha retroazione, cioè è senza feedback. L'azione di controllo del sistema è indipendente dall'uscita che viene controllata. L'uscita non viene misurata né riportata in ingresso. In un sistema ad anello aperto si confida che l'uscita segua ciecamente il comando dato in ingresso indipendentemente dal risultato ottenuto (Figura 12).



→ Fig. 12 Controllo ad anello aperto senza retroazione.

#### Pit Stop

- 12 Come agisce un controllo ad anello aperto?  
 13 In quali circostanze è più indicato usare Raspberry Pi?

## 4 Il Physical Computing con Arduino e Raspberry Pi

Arduino e Raspberry Pi sono tra i sistemi di elaborazione più utilizzati nel Physical Computing.

Sono dotati di porte di ingresso/uscita che permettono di controllare i segnali digitali e analogici con cui interagire con il mondo fisico.

La scelta tra i due sistemi dipende molto dall'applicazione che si vuole realizzare:

- Raspberry Pi è ideale per interagire con gli utenti e con dispositivi standard (per esempio porte USB, Ethernet, Wi-Fi), o per realizzare sistemi che richiedono più capacità di calcolo o funzioni di sistema operativo;
- Arduino è ottimo per operazioni semplici, veloci e a basso consumo energetico.

Entrambi i sistemi dispongono di un ricco corredo hardware (componenti elettronici, schede di espansione, kit di sviluppo) e software open source.

### ■ Segnali analogici e segnali digitali

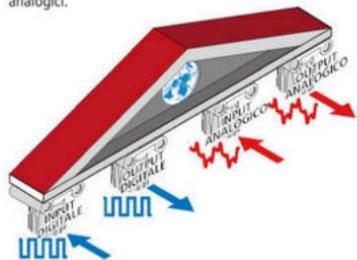
Le applicazioni che per interfacciarsi con il mondo fisico utilizzano sensori e attuatori devono gestire segnali di ingresso/uscita che possono essere digitali/analogici.

Ricordiamo che un segnale digitale (discreto) ha un insieme finito di stati, un segnale analogico (continuo) ha un insieme infinito di stati.

I casi che si possono presentare sono illustrati nella Figura 13.

- Output digitale** (per esempio il blink di un LED);
- Input digitale** (per esempio la lettura di un pulsante);
- Input analogico** (per esempio la lettura di un potenziometro);
- Output analogico** (per esempio la Pulse Width Modulation).

↓ Fig. 13 I quattro pilastri su cui si regge il controllo di sensori e attuatori digitali e analogici.



La Tabella 1 mette a confronto i segnali **digitali** con i segnali **analogici**.

| DIGITALE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | ANALOGICO                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                               |
| <ul style="list-style-type: none"> <li>I calcolatori lavorano con sistemi digitali basati sui valori 0 e 1.</li> <li>I segnali analogici vanno necessariamente convertiti in digitale. Si tratta di metodi che permettono di tradurre un fenomeno analogico in una sequenza di due soli elementi: 1 e 0, on e off, acceso e spento.</li> <li>La ricchezza di un suono viene convertita in una stringa di bit applicando semplici funzioni matematiche che permettono di approssimare numericamente la curva che rappresenta, istante per istante, la pressione dell'aria.</li> <li>Un'immagine viene convertita in pixel, cioè in una serie di punti di cui si memorizza il valore numerico del colore e delle quantità di luce.</li> </ul> | <ul style="list-style-type: none"> <li>I fenomeni naturali (suoni, pressioni, temperature) sono tipicamente analogici.</li> <li>Un segnale analogico è composto di infinite variazioni.</li> <li>Un suono ha infinite variazioni. La sola pressione dell'aria è in grado di trasmettere una stupenda sinfonia, gli infiniti timbri della voce o il rumore di una cascata.</li> <li>Un'immagine possiede infinite variazioni di intensità di colore.</li> </ul> |

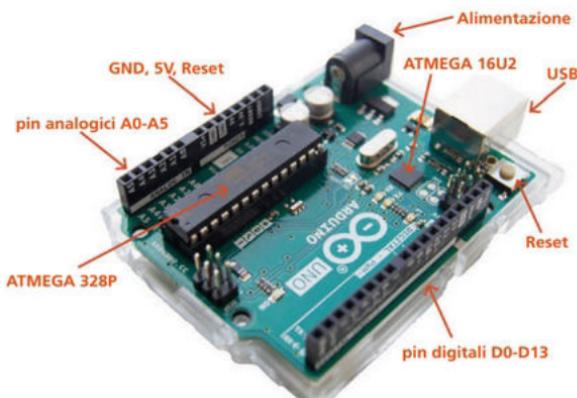
### ■ Arduino e Raspberry Pi: due schede a confronto

↑ Tab. 1 Confronto fra segnali digitali e analogici.

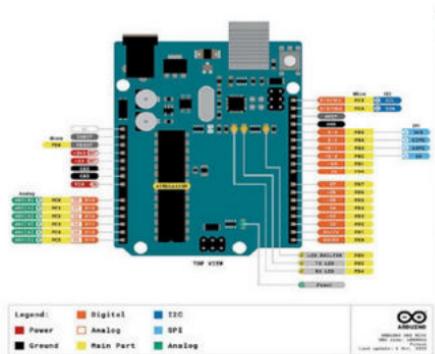
#### Arduino

Arduino è una **scheda hardware** che racchiude un **microcontrollore (MCU)** capace di elaborare le informazioni che arrivano dall'ambiente esterno e di integrare con esso.

Il chip del microcontrollore (**Figura 14a**) integra il processore, la memoria e, tramite le porte di interfaccia (**Figura 14b**), le unità di Input/Output programmabili.



◀ Fig. 14a Arduino UNO R3 con microcontrollore ATmega328u4 prodotto da AVR con architettura Harvard.

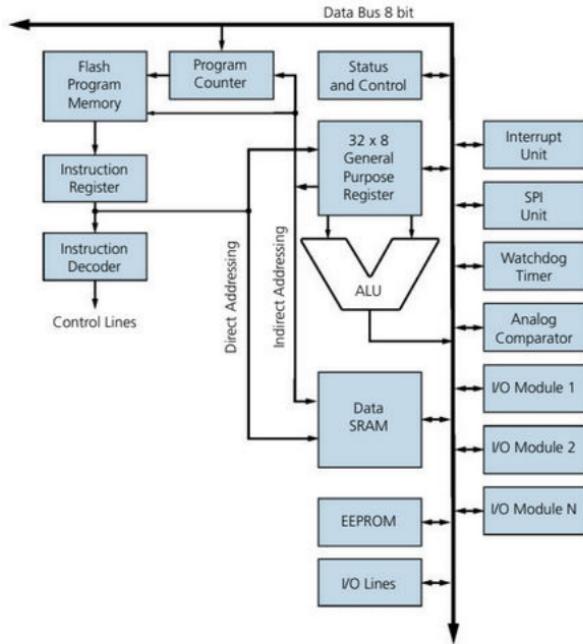


→ Fig. 14b Porte di interfaccia. I pin PWM sono riconoscibili dal simbolo «~» che si legge accanto al numero della porta (pin 3, 5, 6, 10 e 11).

### Pit Stop

- 14 Che cos'è Arduino?  
15 Dove sono archiviati il codice e la memoria?

Arduino utilizza, nella maggior parte dei modelli, i microcontrollori della famiglia **AVR** basati sul modello dell'architettura **Harvard**, in cui il codice del programma e i dati sono in memorie separate: il codice è archiviato nella memoria flash, i dati nella memoria RAM. La CPU è dotata di un set di istruzioni, molte delle quali sono eseguite in un unico ciclo di clock. La **Figura 15** mostra lo schema interno di uno dei microcontrollori di tale famiglia, il microcontrollore Atmel ATmega328P.



→ Fig. 15 Schema interno del microcontrollore Atmel ATmega328P con unità di elaborazione (ALU, Registri, Program Counter, Status Register), memorie (EEPROM, flash, SRAM) e porte di I/O.

Per lo sviluppo del software è disponibile un ambiente di sviluppo integrato (**IDE – Integrated Development Environment**) che permette di scrivere i programmi (*sketch*) in un linguaggio molto vicino al linguaggio C/C++. Uno degli elementi caratterizzanti di Arduino è la facilità con cui è possibile interagire con il mondo esterno, raccogliere segnali provenienti dai sensori e pilotare gli attuatori.

La **Tabella 2** riassume le caratteristiche principali hardware e software di Arduino Uno con le istruzioni per la programmazione dei canali di interfaccia.

| Hardware                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Microcontrollore                   | <ul style="list-style-type: none"> <li>Atmel ATmega328P (famiglia AVR):           <ul style="list-style-type: none"> <li>architettura RISC;</li> <li>32 registri a 8 bit di uso generale;</li> <li>memoria <i>flash</i>: 32 kB;</li> <li>EEPROM: 1kB;</li> <li>RAM: 2 kB (SRAM).</li> </ul> </li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Ingressi/Uscite                    | <ul style="list-style-type: none"> <li>14 pin di ingresso/uscita digitali (di cui 6 utilizzabili come uscite PWM);</li> <li>6 ingressi analogici;</li> <li>porta USB utilizzata per la comunicazione con il computer utilizzato per la stesura e il <i>debug</i> dei programmi.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Software                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Ambiente di sviluppo               | <ul style="list-style-type: none"> <li>IDE (<i>Integrated Development Environment</i>);</li> <li>linguaggio C/C++ specializzato per Arduino.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Gestione dei canali di interfaccia | <ul style="list-style-type: none"> <li>Possono essere impostati in ingresso o in uscita, durante la fase di <i>setup</i> (istruzione <i>pinMode</i>);</li> <li>la lettura e scrittura dei segnali digitali avviene, rispettivamente, con le istruzioni di <i>digitalRead</i> e <i>digitalWrite</i>;</li> <li>i pin 3, pin 5, pin 6, pin 10, pin 11 sono dedicati alla codifica dei «segnali con una modulazione di larghezza di impulso» (PWM – <i>Pulse-Width Modulation</i>) che emette un'onda quadra variabile in funzione del <i>duty cycle</i>, cioè del rapporto tra la durata dello stato ON del segnale rispetto al periodo totale. In questo modo è possibile simulare un comportamento analogico dell'uscita;</li> <li>i canali di ingresso analogici sono collegati a convertitori analogico-digitale (ADC). I valori di tensione continua, compresi tra 0 V e 5 V, sono convertiti in valori numerici, discreti, compresi tra 0 e 1023.</li> </ul> |

◀ **Tab. 2** Caratteristiche tecniche di base di Arduino Uno.

### Raspberry Pi

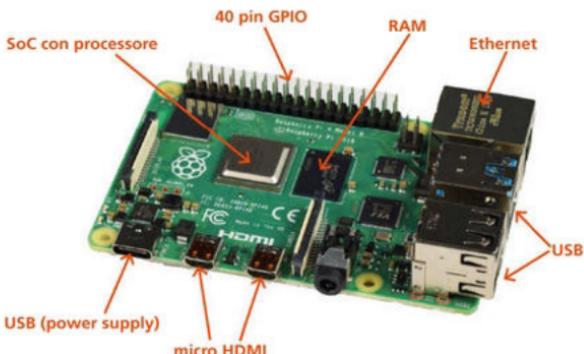
Raspberry Pi è un computer racchiuso in una scheda singola (SBC) che si interfaccia facilmente al mondo esterno. Dal punto di vista **hardware** il funzionamento della scheda (**Figure 16a e 16b**):

- si basa su un SoC, che integra CPU, GPU, RAM e *chipset*;
- per la memoria di massa è previsto un alloggiamento per l'inserimento di una scheda flash SD che contiene il boot e il sistema operativo;
- l'interfaccia di Input/Output (**GPIO – General Purpose Input Output**) consente di accedere facilmente ai pin per la gestione dei segnali esterni. A differenza di Arduino, non ha ingressi analogici, ma è dotato di porte USB, Ethernet e trasmettitore Wi-Fi.

### notabene

Il sito ufficiale fornisce un'ampia e aggiornata documentazione.

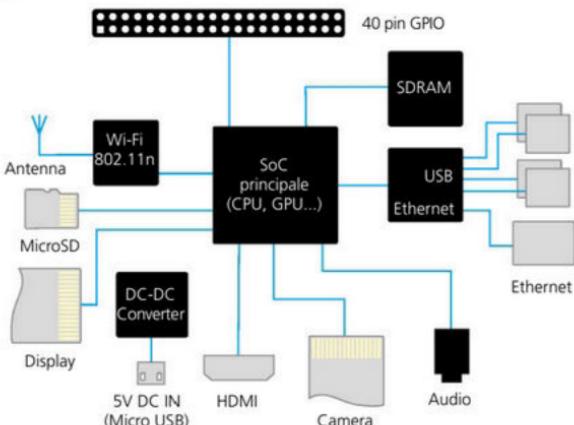
a.



→ Fig. 16a La scheda Raspberry Pi ha le dimensioni di una carta di credito: 82 mm × 56 mm × 19,5 mm, per 50 g di peso.

b Schema a blocchi di base della scheda Raspberry Pi.

b.



Il vero cardine attorno al quale ruotano le applicazioni della scheda è la gestione dell'interfaccia **GPIO** (*General Purpose Input/Output*) che, con i suoi pin, permette di far interagire Raspberry Pi con il mondo esterno. Il connettore GPIO (modello B+) ha 40 pin (Figura 17), che comunicano direttamente con il processore e sono facilmente programmabili in ingresso, per la lettura di sensori, o in uscita, per comandare LED, motori, relè, ecc. Inoltre è possibile utilizzare alcuni pin di GPIO, pensati per dialogare con altre periferiche, utilizzando comunicazioni seriali asincrone e sincrone (UART, I2C, SPI).

La scelta migliore per programmare i pin di GPIO è quella di utilizzare la libreria RPi.GPIO di Python, cosa che faremo nelle attività di laboratorio.



|          |    |           |          |
|----------|----|-----------|----------|
| 3.3V     | 1  | 2         | 5V       |
| GPIO2    | 3  | 4         | 5V       |
| GPIO3    | 5  | 6         | GND      |
| GPIO4    | 7  | 8         | GPIO14*  |
| GND      | 9  | 10        | GPIO15** |
| GPIO17   | 11 | 12        | GPIO18   |
| GPIO27   | 13 | 14        | GND      |
| GPIO22   | 15 | 16        | GPIO23   |
| 3.3V     | 17 | 18        | GPIO24   |
| GPIO10   | 19 | 20        | GND      |
| GPIO9    | 21 | 22        | GPIO25   |
| GPIO11   | 23 | 24        | GPIO8    |
| GND      | 25 | 26        | GPIO7    |
| DNC      | 27 | 28        | DNC      |
| GPIO5    | 29 | 30        | GND      |
| GPIO6    | 31 | 32        | GPIO12   |
| GPIO13   | 33 | 34        | GND      |
| GPIO19   | 35 | 36        | GPIO16   |
| GPIO26   | 37 | 38        | GPIO20   |
| GND      | 39 | 40        | GPIO21   |
| *UART TX |    | **UART RX |          |

◀ Fig. 17 Schema dei pin di GPIO. I pin GPIO hanno funzioni alternative disponibili su pin specifici.

| PIN fisici                                                                                                  | Significato                                                                                                                                                                                                                               |
|-------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 e 17                                                                                                      | Alimentazione 3,3 V (PWR, Power)                                                                                                                                                                                                          |
| 2 e 4                                                                                                       | Alimentazione 5 V (PWR, Power)                                                                                                                                                                                                            |
| 6, 9, 14, 20, 25, 30, 34 e 39                                                                               | Messa a terra (GND, Ground)                                                                                                                                                                                                               |
| 3, 5, 7, 8, 10, 11, 12, 13, 15, 16, 18, 19, 21, 22, 23, 24, 26, 27, 28, 29, 31, 32, 33, 35, 36, 37, 38 e 40 | BCM – Broadcom SOC channel<br>Nota: GPIO 8 e 10 possono essere utilizzati, rispettivamente, come fili di TX e RX per la trasmissione seriale asincrona; altri pin GPIO possono essere usati per trasmissioni seriali sincrone (SPI, I2C). |

Per quanto riguarda il software, Raspberry Pi è stato progettato per ospitare distribuzioni Linux (tra cui **Raspbian**, una distribuzione adattata al suo hardware), ma supporta anche altri sistemi operativi, compreso Windows.

Per sviluppare software su Raspberry Pi si possono utilizzare linguaggi e ambienti di sviluppo diversi: Sonic Pi Write per produrre musica, Scratch per creare animazioni e giochi didattici, C++, Java, PHP, Ruby ecc. Tuttavia, Raspberry Pi Foundation privilegia e supporta **Python**, un linguaggio semplice e completo. La programmazione a oggetti e l'uso di librerie esterne lo rendono versatile e potente. Sono state sviluppate molte versioni di Raspberry, che hanno ampliato e migliorato le prestazioni senza stravolgere la struttura del progetto iniziale e i principi di funzionamento.

La **Tabella 3** a pagina seguente riassume le caratteristiche principali hardware e software di **Raspberry Pi 4 (Model B)**, ma che, nella sostanza, sono comuni ai diversi modelli, anche se i microcontrollori della famiglia ARM, su cui si basano, possono appartenere a famiglie con architettura von Neumann o Harvard.

### notabene

Raspberry Pi è anche molto apprezzato nei progetti di domotica, videogiochi, robotica e applicazioni industriali.

**Pit Stop**

- 16** Che cos'è Raspberry?
- 17** Quali sono le caratteristiche hardware di Raspberry?
- 18** Come si fa a scrivere software per Raspberry?

| Hardware                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SoC                                | Broadcom BCM2711:<br><ul style="list-style-type: none"> <li>CPU quad-core ARM v8, 64-bit;</li> <li>GPU (VideoCore IV 3D);</li> <li>RAM: fino a 8GB di SDRAM.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Ingressi/Uscite                    | <ul style="list-style-type: none"> <li>Porte:           <ul style="list-style-type: none"> <li>– Bluetooth;</li> <li>– Gigabit Ethernet;</li> <li>– USB.</li> </ul> </li> <li>Interfaccia GPIO con 40 pin (<b>Figura 16</b>) che comprende anche:           <ul style="list-style-type: none"> <li>– un bus seriale I2C;</li> <li>– PWM (modulatore di larghezza di impulso);</li> <li>– UART, trasmettitori ricevitori asincroni universali.</li> </ul> </li> <li>HDMI.</li> <li>Camera.</li> <li>Sistemi audio e video.</li> <li>Slot Micro-SD per l'inserimento della card per il salvataggio del sistema operativo, dati e programmi.</li> </ul> <p>Si deve notare che Raspberry Pi non dispone di un ingresso analogico. Occorre quindi montare un convertitore analogico-digitale (per esempio MCP3002 – 2.7 V Dual Channel 10-Bit A/D Converter), oppure utilizzare Arduino come «sensore» e collegarlo via seriale a Raspberry Pi.</p> |
| Software                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Ambiente di sviluppo               | <ul style="list-style-type: none"> <li>Raspbian (sistema operativo ufficiale basato su Linux Debian);</li> <li>Windows 10 IoT (una versione speciale di Windows per Raspberry Pi);</li> <li>linguaggio: Python.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Gestione dei canali di interfaccia | <p>Metodi per definire una linea come ingresso o come uscita e controllarne lo stato:</p> <ul style="list-style-type: none"> <li>libreria GPIO (<i>import RPi.GPIO</i>);</li> <li>modalità di numerazione dei pin su GPIO (<i>setmode(GPIO.BCM)</i>);</li> <li>impostazione (funzioni di setup, in Input o in Output);</li> <li>scrittura e lettura dei pin (<i>GPIO.output</i> e <i>GPIO.input</i>).</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

→ **Tab. 3** Caratteristiche tecniche di base di Raspberry Pi 4 (Model B).

## Il sogno di realizzare un computer in cui ogni studente potesse «mettere le mani»

Nel 2012 la Raspberry Pi Foundation presentò Raspberry Pi come parte di un progetto open source inteso a «promuovere lo studio dell'informatica e riportare uno spirito di divertimento nell'apprendimento del computer».

Il sogno di un **sistema economico, semplice e aperto** era nato un paio di anni prima, all'Università di Cambridge, dove Eben Upton e i suoi collaboratori decisero di realizzare un computer in cui ogni studente avrebbe potuto «mettere le mani», per programmarlo in modo divertente e collegarlo con facilità ai dispositivi esterni. Per la realizzazione si utilizzò il SoC (*System on a Chip*) di Broadcom, dotando il sistema di una porta video (per utilizzare anche vecchi televisori come monitor) e installando sistema operativo e applicazioni open source gratuiti (per non creare barriere e disporre di una comunità in espansione).

Alla fine si ottenne un sistema di basso costo, affidabile e versatile e con prestazioni di tutto rispetto: un successo!

Non solo Raspberry Pi è utilizzato per le applicazioni più diverse, ma la stessa Microsoft ha adeguato una versione di Windows 10 per farla «girare» sulla scheda.

# Approfondimento - Tecnologia

## Il controllo PID (Proporzionale, Integrale, Derivativo)

Il regolatore PID mantiene il valore di uscita desiderato mediante tre azioni:

- **proporzionale** al valore del segnale di errore;
- **integrale**, cioè che considera i valori storici del segnale di errore;
- **derivativa**, cioè che considera la velocità con cui il segnale di errore cambia.

### Azione Proporzionale (P)

Il regolatore basa le proprie decisioni sulla differenza che esiste tra l'uscita misurata e l'uscita desiderata. La correzione che viene fornita al sistema è proporzionale all'errore riscontrato:

$$\text{correzione} = K \times \text{errore}$$

dove  $K$  è un parametro che viene impostato in funzione della risposta che si vuole ottenere. Di fatto,  $K$  è la pendenza della retta che rappresenta il grafico della funzione «correzione».

Questo tipo di controllo purtroppo ha il difetto che, cercando di contrastare l'errore, il sistema può diventare instabile e oscillare intorno al valore desiderato. È come se, nel guidare un'automobile, per non andare troppo a destra si girasse troppo a sinistra e poi ancora a destra e a sinistra ([Figura 1](#)).



Valore di riferimento    [Fig. 1](#)

### Azione Integrale (I)

Il regolatore utilizza i dati storici per capire per quanto tempo il sistema ha deviato dall'obiettivo: più a lungo il sistema si discosta dal valore impostato, maggiore è la correzione effettuata dal regolatore. L'azione integrale consiste nell'effettuare la somma di tutti gli errori nel tempo; ricordando la storia degli errori, il regolatore non reagisce immediatamente all'errore, rendendo così più stabile il sistema, ma rendendolo anche, d'altra parte, più lento e meno reattivo ([Figura 2](#)).



Valore di riferimento    [Fig. 2](#)

### Azione Derivativa (D)

Il regolatore tiene conto della velocità con cui il sistema si avvicina all'obiettivo desiderato. Se il segnale di errore sta variando velocemente, si cerca di intervenire, senza aspettare che l'errore diventi significativo e che duri per troppo tempo. Talvolta, tuttavia, l'azione derivativa rende il controllo troppo sensibile e, per questo, non sempre è consigliata.

### Il regolatore PID

È la semplice **somma delle tre azioni** ( $\text{PID} = \text{P} + \text{I} + \text{D}$ ), che tende a far arrivare il sistema a regime in modo ottimale ([Figura 3](#)).



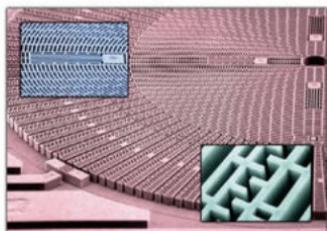
Valore di riferimento    [Fig. 3](#)

# Approfondimento - Tecnologia

ORIENTAMENTO

S T E M

## Sensori e MEMS



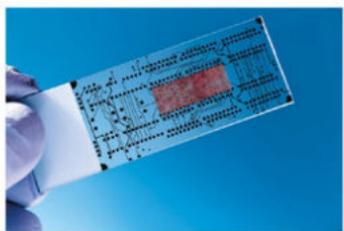
↑ Fig. 1 Dettaglio di un sensore inerziale di STMicroelectronics. Mediante uno specifico processo produttivo (bulk micromachining) si utilizza la luce per scavare il silicio in modo da realizzare strutture tridimensionali. Al contrario, attraverso la tecnologia surface micromachining si incide il wafer di silicio per costruirvi sopra dei livelli stratificati. Fonte: STMicroelectronics.

La **nanoelettronica** è una delle tecnologie più pervasive e ha un'enorme influenza sulla nostra vita quotidiana, sul nostro modo di lavorare, di divertirci e di comunicare. Ha alimentato e continuerà ad alimentare nel futuro lo sviluppo dell'informatica, dell'elettronica di consumo, delle telecomunicazioni e del settore industriale (basti pensare a quanta micro- e nanoelettronica ci sia oggi in un'automobile).

Tradizionalmente la **microelettronica** si occupava solo della parte centrale degli elaboratori, mentre la componentistica di altro tipo (elettromeccanica, tubi a vuoto, dispositivi discreti di potenza) si occupava delle interfacce verso il mondo fisico; ormai la microelettronica, o quantomeno l'infrastruttura tecnologica basata sul silicio, ha iniziato a presidiare pesantemente anche questa parte dei sistemi. Da circa vent'anni, nella microelettronica coesistono due filoni di sviluppo: uno continua a perseguire la miniaturizzazione e l'aumento di densità come sua forza trainante, ed è chiamato «**more of Moore**», l'altro, chiamato «**more than Moore**», sfrutta le tecnologie microelettroniche per implementare funzioni diverse.

Una delle più importanti tecnologie del filone *more than Moore* è quella dei **MEMS** (*Micro Electro Mechanical Systems*, sistemi micro-elettro-mecanici). Oltre alle note proprietà elettriche, il silicio ha infatti anche ottime proprietà meccaniche; l'infrastruttura tecnologica della microelettronica consente di sfruttare queste proprietà per realizzare sistemi meccanici su scala micrometrica e permette anche di integrare o interfacciare direttamente questi microsistemi con l'elettronica di controllo.

Si possono realizzare per esempio oggetti come quello mostrato in Figura 1, molto apprezzabile anche dal punto di vista estetico.



↑ Fig. 2 Lab on Chip (LOC) è un dispositivo che integra le funzioni di laboratorio su un nanochip: il chip contiene un vero e proprio laboratorio in miniatura, in cui migliaia di reazioni possono essere eseguite in parallelo. I dispositivi LOC appartengono alla famiglia dei MEMS.

È un esempio di **sensore inerziale**, in cui ci sono due parti interdigitate, una fissa e l'altra sospesa; in presenza di un'accelerazione angolare, la distanza relativa tra le due parti si modifica a causa dell'inerzia della parte sospesa. La variazione della distanza tra le due parti può essere rilevata misurando la capacità elettrica della struttura. In questo modo si realizza un **sensore di rotazione**.

Grazie a sensori inerziali MEMS, le console dei videogiochi consentono all'utente di giocare, simulando movimenti reali. Gli smartphone e i tablet sono dotati di sensori inerziali che consentono di mantenere verticale l'immagine sullo schermo

quando si ruota l'apparecchio.

Giroscopi micromecanici permettono ai telefoni cellulari di non perdere la localizzazione quando si cammina all'interno degli edifici.

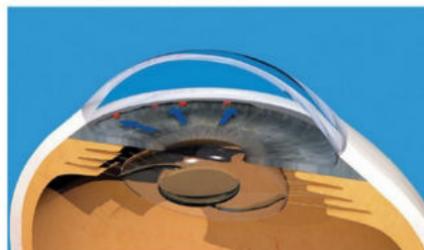
La tecnologia MEMS, basata sulle nanotecnologie, consente inoltre di integrare funzioni elettroniche, meccaniche, ottiche, biologiche o chimiche in spazi ridottissimi in un unico componente.

Un dispositivo MEMS contiene, su un minuscolo chip di silicio, microcircuiti integrati con dispositivi di tipo meccanico o sensori (Figura 2).

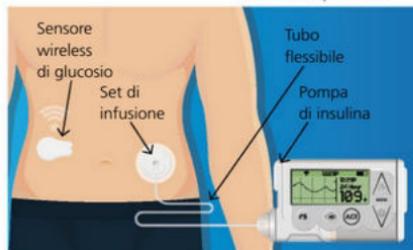
In questo modo è possibile disporre di **sensori microelettromeccanici**: microfoni, giroscopi, magnetometri, accelerometri, bussole digitali, sensori per ultravioletti, sensori tattili, rilevatori di pressione e di umidità.

Altre applicazioni interessanti dei MEMS si hanno nella **robotica**, dove si fa largo uso di giroscopi e accelerometri per determinare la posizione nello spazio del sistema, la sua accelerazione o inclinazione. Poder controllare il movimento di un sistema, compresa la sua accelerazione o inclinazione, permette di ottenere applicazioni intuitive, realistiche, interattive e totalmente integrate (*embedded*). In **campo automobilistico** possiamo trovare i MEMS ovunque: airbag, sistemi di navigazione assistita, scatole nere. I MEMS giocano un ruolo fondamentale nei sistemi di **antifurto**, proprio perché in grado di rilevare gli spostamenti. Nel campo industriale e dei beni di consumo (per esempio gli elettrodomestici) i MEMS possono essere utilizzati per rilevare vibrazioni anomale e individuare usure nelle parti meccaniche.

↓ Fig. 3 L'applicazione dei MEMS al corpo umano come biosensori per la rilevazione della pressione oculare (a.), o per la rilevazione del glucosio e la somministrazione dell'insulina con le pompe di insulina (b.), aprono nuovi scenari in campo medico.



a.



b.

Anche in **ambito medico** i MEMS trovano vaste applicazioni. Per esempio, vengono usati per il controllo della pressione oculare in patologie quali il diabete (Figura 3a), oppure nelle protesi per il movimento degli arti e per la somministrazione intelligente dell'insulina (Figura 3b).

Un'altra applicazione di successo delle tecnologie *more than Moore* è l'**imaging**. Anche le macchine fotografiche di alta gamma sono infatti dotate di sensori digitali realizzati su silicio; i sensori delle fotocamere sono matrici di transistor e condensatori, coperte da strati di materiali dielettrici trasparenti, opportunamente lavorati, che fanno da filtro di colore e da lente per ciascun pixel. Nelle videocamere è inoltre possibile compensare il movimento e stabilizzare le immagini. Nei proiettori di nuova generazione si trovano infine applicazioni per la gestione di microspecchi.

Numerose **app didattiche** per lo studio delle scienze sfruttano in modo accattivante e divertente i sensori interni degli smartphone (Figura 4). Le prospettive e i vantaggi sono enormi, anche se devono tenere presenti alcuni problemi legati a questi dispositivi, come il fatto di dover trattare segnali analogici di bassissima energia, con una differenza minima tra segnale e rumore. Anche la progettazione e la produzione di questi microsensori presenta problemi, per esempio legati a stress termici o di altra natura.

La produzione dei MEMS in grandi quantità permette la riduzione dei costi e soprattutto, nei prossimi anni, li renderà particolarmente convenienti e versatile, aprendo scenari tutti ancora da esplorare.



↑ Fig. 4 L'app Science Journal di Google consente di utilizzare i sensori interni dello smartphone per misurare e monitorare luce, suono, movimento, campi magnetici.

# Laboratorio

## Disegnare gli schemi di sistemi basati su controlli ad anello aperto e chiuso

Un sistema controllato ad anello aperto prevede che si ordini all'impianto di eseguire una determinata azione senza verificarne i risultati. Un sistema ad anello chiuso invece controlla costantemente l'uscita e regola l'ingresso per conseguire il risultato desiderato.

### ATTIVITÀ 1

#### Controllo ad anello aperto

##### Scenario

Nel caso in cui si voglia utilizzare il forno di casa per cucinare le patate e non si abbia pratica di cucina, si segue una ricetta che spiega quali sono la temperatura e il tempo di cottura. Per esempio: «Dopo aver lavato e tagliato a spicchi le patate, preriscaldate il forno a 180 °C tenendovi all'interno la teglia che servirà per cuocere le patate. Oliate la teglia e sistemate le patate nella teglia. Cuocete in forno a 200 °C per mezz'ora. Terminata la cottura, sfornate e servite!»

##### Consegna

Disegnare lo schema per cucinare le patate con il forno di casa.

##### Svolgimento

###### **Costruzione dello schema a blocchi del sistema di controllo ad anello aperto**

Il sistema di controllo elabora la condizione di ingresso e produce un'uscita determinata. In questo caso la temperatura del forno viene impostata a 200 °C e ci si aspetta che dopo mezz'ora si apra il forno e le patate siano cotte (Figura 1):



→ Fig. 1  
Schema per  
cucinare le  
patate al forno

Altri scenari possibili sono:

- potare una siepe;
- concimare l'orto;
- scrivere al cellulare (o suonare il piano) senza guardare la tastiera;
- prescrivere una cura medica dopo una diagnosi;
- fare le scale (per esempio quando si esce all'ultima ora da scuola...) senza guardare i gradini.

### ATTIVITÀ 2

#### Controllo ad anello chiuso

##### Scenario

Consideriamo un semplice serbatoio, come quello illustrato nella Figura 2, che ha una valvola di riempimento per controllare il livello del liquido. Quando il livello nel serbatoio aumenta, il galleggiante si solleva provocando la chiusura della valvola che porta il liquido al serbatoio.

## Consegna

Disegnare lo schema del funzionamento del serbatoio.

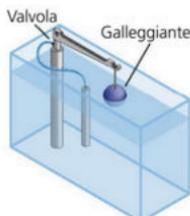


Fig. 2 Il dispositivo a valvola per il controllo del livello del liquido in un serbatoio.

## Svolgimento

### Costruzione dello schema a blocchi del sistema di controllo ad anello chiuso

Il sistema di controllo ad anello chiuso (Figura 3) monitora l'uscita e interviene sugli ingressi. Il sensore fornisce la misura dell'uscita con le informazioni di feedback utili alla regolazione del processo.

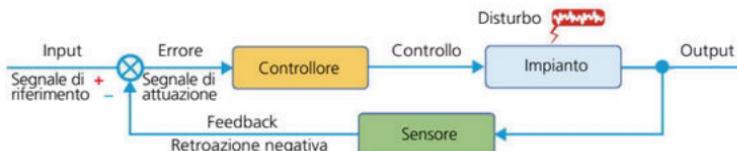


Fig. 3 Sistema di controllo ad anello chiuso con i suoi componenti.

Il controllo di riempimento del serbatoio può essere rappresentato come un sistema a retroazione negativa (Figura 4):

- l'ingresso è costituito dall'afflusso del liquido;
- l'elemento di controllo è la valvola;
- l'impianto è il serbatoio;
- l'uscita è il livello del liquido;
- il sensore è il galleggiante;
- il percorso di retroazione avviene attraverso il braccio della leva della valvola;
- il segnale di attuazione è la posizione della valvola.

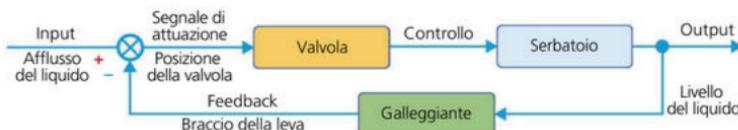


Fig. 4 Il diagramma del dispositivo per il controllo del livello del liquido in un serbatoio è un semplice esempio di apparecchiatura a retroazione negativa.

Altri scenari possibili sono:

- un termostato d'ambiente;
- un forno industriale con controllo a circuito chiuso automatizzato con sensore di temperatura retroazionato;
- la guida dell'auto, della bici, della motocicletta...;
- salire i gradini (per esempio quando si entra a scuola...);
- monitorare un paziente in sala di rianimazione;
- ormeggiare una barca a una boa.

## Progettare con Arduino

Per realizzare un sistema utilizzando la scheda Arduino è necessario dividere il **progetto** in due parti:

1. hardware;
2. software.

Per la progettazione della parte **hardware** si possono usare i kit che contengono varie componenti: breadboard, LED, potenziometri di varia tipologia, push button, fotoresistenze, sensori di temperatura, servomotori, buzzer, relay, LCD ecc. Le schede di espansione (*shield*) sono spesso utilizzate per la gestione di sensori e attuatori di tutti i tipi: dai motori ai collegamenti wireless, dai dispositivi per videogiochi ai sistemi di geolocalizzazione.

Per lo sviluppo del **software** si utilizza un ambiente di sviluppo integrato IDE (*Integrated Development Environment*) che permette di scrivere, compilare e caricare programmi, tramite la porta USB, sulla scheda Arduino.

L'IDE può essere scaricato dal sito ufficiale di Arduino ed eseguito su un normale PC con sistema operativo Linux o Windows, oppure può essere utilizzata la versione on line.

Un **programma per Arduino** è chiamato *sketch* ed è scritto con un linguaggio simile al C. La struttura base di uno sketch prevede due sezioni (*funzioni*):

- *setup()*, eseguita una volta sola e utilizzata per l'inizializzazione e l'impostazione di parametri, variabili e funzioni di libreria;
- *loop()*, il cui codice è ripetuto all'infinito.

In alcuni casi può essere utile utilizzare un simulatore. I vantaggi sono evidenti: solleva il progettista dai rischi e dalle difficoltà legate all'attività pratica, riduce sensibilmente i costi di sviluppo del progetto, consente di poter vedere la soluzione così come accadrebbe nella realtà. Di seguito utilizzeremo:

- **Tinkercad**<sup>®</sup>, un programma di modellazione 3D che permette anche di progettare e simulare circuiti ed eseguire programmi di Arduino. Il software è di proprietà di Autodesk<sup>®</sup>;
- **Fritzing**, uno strumento software, libero, molto utile per chi desidera progettare, esportare e produrre circuiti stampati (PCB, *Printed Circuit Board*) raggiungibile all'indirizzo <https://fritzing.org>;
- **Packet Tracer**, di proprietà CISCO, che simula il comportamento di una rete e dei suoi componenti.

## ATTIVITÀ 3

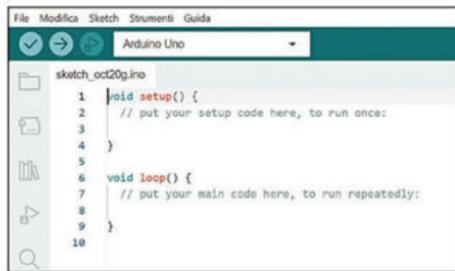
### Utilizzo di IDE per lo sviluppo del software

#### Consegna

Utilizzare l'IDE di Arduino per scrivere uno sketch.

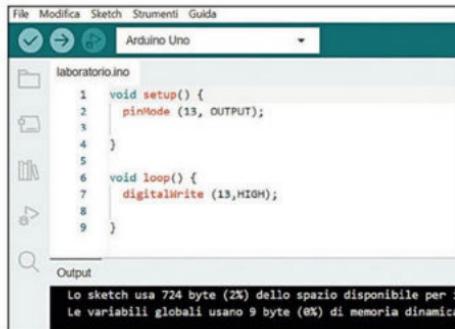
#### Svolgimento

**Passo 1:** installare e aprire IDE di Arduino. Selezionare File → New Sketch per creare un nuovo programma.



```
File Modifica Sketch Strumenti Guida
Arduino Uno
sketch_ocl20g.ino
1 void setup() {
2     // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8
9 }
10
```

**Passo 2:** scrivere il codice e salvare il file con estensione .ino



```
File Modifica Sketch Strumenti Guida
Arduino Uno
laboratorio.ino
1 void setup() {
2     pinMode (13, OUTPUT);
3
4 }
5
6 void loop() {
7     digitalWrite (13,HIGH);
8
9 }
```

Output

```
Lo sketch usa 724 byte (2%) dello spazio disponibile per i
le variabili globali usano 9 byte (0%) di memoria dinamica
```

**Passo 3:** verificare la correttezza del codice, compilarlo e caricarlo utilizzando i tasti .

**Passo 4:** collegare il cavo USB alla scheda.

Nella sezione **Strumenti (Tools)** impostare con le caratteristiche desiderate la board e la porta seriale utilizzata.

È possibile utilizzare il Monitor Seriale per effettuare il debug.

```

File Modifica Sketch Strumenti Aiuto
newPhotoResistor_p.ino [RECENTI]
void setup() {
  pinMode(ledPin, OUTPUT); // LED ->OUT
  Serial.begin(9600);
}

void loop() {
  int lighIntensity = analogRead(analogPin); // legge tensione analogica
  Serial.print("Intensità Luce: ");
  Serial.print(lighIntensity);
  Serial.print("\n");
}

```

The serial monitor shows the following data:

```

COM3 Invia
Intensità Luce: 0 0
Intensità Luce: 32 7
Intensità Luce: 04 20
Intensità Luce: 73 18
Intensità Luce: 0 1
Intensità Luce: 0 0
Intensità Luce: 31 7
Intensità Luce: 85 21
Intensità Luce: 73 18

```

## ATTIVITÀ 4

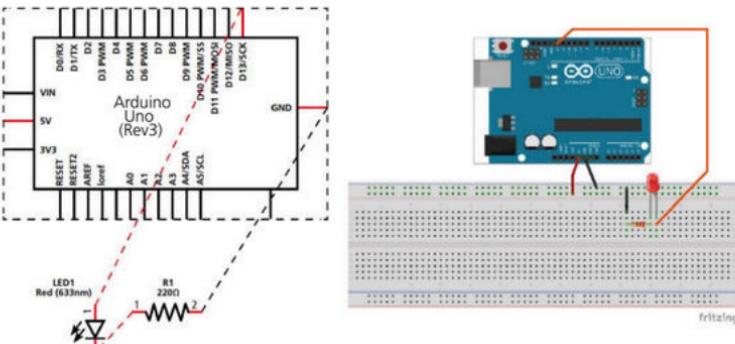
### Output digitale: lampeggi di un LED

#### Consegna

Facciamo lampeggiare un LED.

#### Svolgimento

Il più classico degli esercizi consiste nel pilotare ciclicamente un'uscita digitale per l'accensione di un LED montato sulla breadboard esterna. Il LED è collegato al pin 13 (Figura 5), che è inizializzato in OUT (funzione di setup). Il LED viene alternativamente acceso e spento, per 2 s (2000 ms) e per 1 s rispettivamente. L'utilizzo della costante `LED _ 13` non è strettamente necessario, ma rende più intuitivo il programma.



↑ Fig. 5 Lo schema elettrico e quello pratico del lampeggio del LED. Gli schemi sono stati realizzati utilizzando con Fritzing un software libero utilizzato per la progettazione di semplici prototipi, basati su breadboard.