

# Sommario

## 1 Architettura dell'elaboratore

### UNITÀ 1:

#### Il sistema di elaborazione e la CPU

1	Introduzione	4
2	I sistemi	4
	Comprendi con l'analogia In aereo di notte: pensare a un sistema e alle sue parti	5
3	Il sistema di elaborazione	5
4	L'architettura di von Neumann	6
	Comprendi con l'analogia L'architettura di von Neumann è come le strade di Dodge City	7
5	Il bus di sistema	7
	Comprendi con l'analogia Un bus a $n$ bit è come un'autostrada a $n$ corsie	8
6	Le componenti della CPU	9
7	L'Unità di Controllo (CU)	10
8	I registri	10
9	L'Unità Aritmetico-Logica (ALU)	11
10	Il clock	12
	Comprendi con l'analogia Il clock è come il capovoglia che nelle antiche triremi scandiva il ritmo di voga	12
11	Esecuzione di un'istruzione	13
	• Sintesi	14
	• Verifica delle conoscenze	16
	• Verifica delle abilità	17
	• CLIL The computing system	18

#### Risorse digitali - Unità 1

1	Approfondimento: Gli Automi	4
2	Mappa interattiva	15
3	Esercizi interattivi	16
4	Esercizi in più	17
5	Audio sintesi in inglese	18
6	Esercizi interattivi in inglese	18

### UNITÀ 2: La memoria

1	Introduzione: memoria primaria e memorie secondarie	20
	Comprendi con l'analogia La memoria primaria è come la memoria a breve termine del nostro cervello, mentre quella secondaria è come la nostra memoria a lungo termine	20
2	Capacità della memoria	20

#### Comprendi con l'analogia

Una memoria finita è come un foglio a quadretti, un quadretto è come un bit	22	
3	La memoria primaria o centrale	22
4	L'indirizzo di memoria	23

#### Comprendi con l'analogia

La memoria è come una cassetiera; le locazioni di memoria sono come i cassetti; l'indirizzo di una locazione è come il numero di un cassetto	23
Comprendi con l'analogia L'indirizzo di una locazione di memoria è come la numerazione della pagina di un libro	24

5	La decodifica degli indirizzi	24
---	-------------------------------	----

6	Il bus di sistema e l'interazione tra CPU e RAM	27
---	---	----

7	Ciclo di lettura e scrittura	28
---	------------------------------	----

8	Considerazione sulla dimensione dei bus	30
---	---	----

9	Esecuzione di un programma	30
---	----------------------------	----

10	Esecuzione dell'istruzione	32
----	----------------------------	----

11	Lo stack	32
----	----------	----

Comprendi con l'analogia Lo stack è come una pila di libri	33
--	----

12	Operazioni sullo stack	33
----	------------------------	----

13	Le memorie a semiconduttore	35
----	-----------------------------	----

14	La RAM	36
----	--------	----

Comprendi con l'analogia L'accesso diretto e l'accesso sequenziale possono essere paragonati a due modi diversi di utilizzare il telecomando della TV	36
---	----

15	La cache memory	38
----	-----------------	----

16	La ROM	39
----	--------	----

17	Confronto tra le memorie a semiconduttore	40
----	---	----

18	Le memorie di massa	40
----	---------------------	----

19	La gerarchia delle memorie	42
----	----------------------------	----

Comprendi con l'analogia La gestione delle informazioni nella gerarchia delle memorie è come la gestione del processo domestico di rifornimento dell'acqua da bere	43
--	----

20	I principi di località	43
----	------------------------	----

21	L'algoritmo LRU ( <i>Least Recently Used</i> )	44
----	--	----

Comprendi con l'analogia La gestione della memoria con la politica LRU è simile alla gestione di un armadio di giacche pieno da cui, per far posto, si toglie la giacca che non si mette da più tempo	44
---	----

22	Il controllo degli errori di memoria	44
----	--------------------------------------	----

• Sintesi	46
-----------	----

• Verifica delle conoscenze	48
-----------------------------	----

• Verifica delle abilità	49
--------------------------	----

• CLIL The memory	50
-------------------	----

## Risorse digitali - Unità 2

	Approfondimento: I semiconduttori	35
	Videointervista: I semiconduttori	35
	Approfondimento: Metodi di gestione della cache memory	38
	Mappa interattiva	47
	Esercizi interattivi	48
	Esercizi in più	49
	Audio sintesi in inglese	50
	Esercizi interattivi in inglese	50

## UNITÀ 3: Le periferiche di I/O

1	Introduzione	51
	<b>Comprendi con l'analogia</b> I tempi della realtà esterna rispetto a quelli di un computer sono come quelli di un bradipo rispetto ai nostri	52
2	Le periferiche	52
3	Le interfacce	53
4	L'indirizzamento	54
5	La trasmissione dei dati	56
6	Tecniche per la gestione delle periferiche	58
	<b>Comprendi con l'analogia</b> L'interrupt è come l'interruzione dell'attività di un cuoco che esegua una ricetta di cucina	59
	<b>Approfondimento - Tecnologia</b>	61
	Un esempio di gestione degli interrupt	61
	<b>Comprendi con l'analogia</b> La linea dedicata NMI è come una linea telefonica con un numero per le emergenze, la linea INT è come una linea telefonica con un numero per le informazioni	62
•	<b>Sintesi</b>	63
•	<b>Verifica delle conoscenze</b>	64
•	<b>Verifica delle abilità</b>	65
•	<b>CLIL The Input/Output devices</b>	66

## Risorse digitali - Unità 3

	Mappa interattiva	63
	Esercizi interattivi	64
	Esercizi in più	65
	Audio sintesi in inglese	66
	Esercizi interattivi in inglese	66

## UNITÀ 4:

### Il software (con DuplOne)

1	Introduzione	68
2	Che cos'è un programma	68
3	Linguaggi a basso e ad alto livello	70
	<b>Comprendi con l'analogia</b> I codici di un computer sono come quelli della natura	72
4	Dal codice sorgente al codice eseguibile	73
5	Il linguaggio Assembly	74
6	Il microprocessore DuplOne e il suo linguaggio Assembly	74
7	Il set di istruzioni del linguaggio Assembly DuplOne	79
8	Le categorie di istruzioni del linguaggio Assembly DuplOne	82
9	Le strutture di controllo	86
10	Metodi di indirizzamento	87

#### Approfondimento - Tecnologia

La programmazione dei sistemi di Intelligenza Artificiale e l'importanza dei dati che mandiamo in rete

89

#### Laboratorio

•	<b>ATTIVITÀ 1</b> Tradurre nel linguaggio Assembly DuplOne istruzioni scritte nel linguaggio naturale	91
•	<b>ATTIVITÀ 2</b> Utilizzo del linguaggio Assembly DuplOne	94
•	<b>ATTIVITÀ 3</b> Dalla pseudo codifica al linguaggio macchina	94
•	<b>ATTIVITÀ 4</b> Utilizzo dei flag del registro di stato	97
•	<b>ATTIVITÀ 5</b> Dal linguaggio ad alto livello al linguaggio Assembly, fino al linguaggio macchina	98

#### • Sintesi

•	<b>Verifica delle conoscenze</b>	99
•	<b>Verifica delle conoscenze</b>	100
•	<b>Verifica delle abilità</b>	101
•	<b>CLIL The Assembly Software</b>	102

## Risorse digitali - Unità 4

	DuplOne Assembly Simulator	90
	Mappa interattiva	99
	Esercizi interattivi	100
	Esercizi in più	101
	Audio sintesi in inglese	102
	Esercizi interattivi in inglese	102

## UNITÀ 5: Evoluzione dei microprocessori

1	Introduzione	104
2	Prestazioni di un sistema a microprocessore	104
3	Overclocking	106
4	Il progresso delle architetture dei sistemi di elaborazione	107
5	Pipeline	108
	Comprendi con l'analogia Il pipeline è come una sessione d'esame	109
6	Multi-Core	112
7	Due diverse filosofie di progettazione delle architetture: CISC e RISC	112
	Comprendi con l'analogia Le architetture CISC e RISC possono essere comprese con i mattoncini delle costruzioni	113
8	Architetture alternative nel gestire con la memoria dati e istruzioni	113
	Approfondimento - Tecnologia L'evoluzione dei calcolatori: dalle valvole ai computer quantistici	115
	Approfondimento - Tecnologia I computer quantistici	118
• Sintesi		119
• Verifica delle conoscenze		120
• Verifica delle abilità		121
• CLIL CPU performance		122

## Risorse digitali - Unità 5

▶	La nanoelettronica	117
📍	Mappa interattiva	119
📝	Esercizi interattivi	120
💻	Esercizi in più	121
🔊	Audio sintesi in inglese	122
📞	Esercizi interattivi in inglese	122

## Verifica delle competenze

Orientamento al lavoro Microprocessori

Sfida finale Un computer per il gaming

## Lavoriamo con l'Intelligenza artificiale

Un nuovo strumento: l'Intelligenza Artificiale in Internet  
Interrogiamo l'Intelligenza Artificiale

## 2 Utilizzo dei sistemi di elaborazione

### UNITÀ 6: Ambienti di elaborazione

1	Introduzione	136
2	Il Personal Computer	136
3	Il case	136
4	L'alimentatore	137
5	La scheda madre	137
6	La CPU	138
7	Il chipset	139
8	La connessione alle periferiche interne	140
9	La connessione alle periferiche esterne	140
10	Le memorie presenti sulla scheda madre	141
11	Il BIOS e l'avvio del sistema	142
	Comprendi con l'analogia Il BIOS opera come la parte inconscia del nostro cervello al risveglio	143
12	Le schede grafiche e la GPU	143
13	Il microcontrollore: un elaboratore in un chip	144
14	Il SoC: una scheda madre in un chip	144
15	SBC: un computer in una scheda	147
	Laboratorio	148
	ATTIVITÀ 1 Montaggio passo per passo delle parti di un PC desktop	148
• Sintesi		150
• Verifica delle conoscenze		152
• Verifica delle abilità		153
• CLIL Inside the Personal Computer		154

## Risorse digitali - Unità 6

▶	Evoluzione del sistema di calcolo	136
📍	Montaggio del computer	148
📍	Mappa interattiva	151
📝	Esercizi interattivi	152
💻	Esercizi in più	153
🔊	Audio sintesi in inglese	154
📞	Esercizi interattivi in inglese	154

## UNITÀ 7: L'architettura del microprocessore 8086

1	Introduzione	156
2	Architettura 8086	156

<b>Comprendi con l'analogia</b> La CPU e l'unità di prefetch collaborano così come fanno un muratore e un manovale	157	Esercizi in più	199
<b>3 I registri</b>	158	Audio sintesi in inglese	200
<b>4 L'organizzazione della memoria</b>	160	Esercizi interattivi in inglese	200
<b>Comprendi con l'analogia</b> 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	161		
<b>5 I pin della CPU 8086</b>	164		
<b>6 Il multiplexaggio temporale di dati/indirizzi</b>	164		
<b>7 Il software dell'architettura 8086: istruzioni e indirizzamenti</b>	166		
<b>Comprendi con l'analogia</b> La gestione dello stack è simile a quella del portavasoi di una mensa	179		
<b>Approfondimento - Tecnologia</b>			
Trasferimento delle parole 8086	180		
<b>Laboratorio</b>	182		
<b>ATTIVITÀ 1</b> Utilizzare un ambiente di sviluppo per programmi Assembly 8086	182		
<b>ATTIVITÀ 2</b> Impariamo a leggere il contenuto della memoria e dei registri della CPU	185		
<b>ATTIVITÀ 3</b> Esempio con variabili e lettura della symbol table e della memoria	187		
<b>ATTIVITÀ 4</b> Programmazione Assembly 8086: calcolo di una media	189		
<b>ATTIVITÀ 5</b> Programmazione Assembly 8086: confronto di variabili	191		
<b>ATTIVITÀ 6</b> Programmazione Assembly 8086: conversione di una stringa di caratteri ASCII in maiuscolo e in minuscolo	192		
<b>ATTIVITÀ 7</b> Programmazione Assembly 8086: il Crivello di Eratostene	194		
<b>Sintesi</b>	197		
<b>Verifica delle conoscenze</b>	198		
<b>Verifica delle abilità</b>	199		
<b>CLIL 8086 Architecture</b>	200		
<b>Risorse digitali - Unità 7</b>			
Approfondimento: Altre istruzioni 8086	176		
Approfondimento: Esempi di programmazione in Assembly 8086	182		
Problema guidato: Gestione dello stack	182		
Mappa interattiva	197		
Esercizi interattivi	198		
<b>Esercizi in più</b>			
Audio sintesi in inglese	200		
Esercizi interattivi in inglese	200		
<b>UNITÀ 8: L'architettura ARM</b>	201		
<b>1 Introduzione</b>	202		
<b>2 L'evoluzione dei processori ARM</b>	202		
<b>3 L'architettura della CPU ARM a 32 bit</b>	203		
<b>4 I registri</b>	204		
<b>5 Modalità di funzionamento del processore</b>	204		
<b>6 La memoria</b>	206		
<b>7 Il software ARM</b>	207		
<b>8 Come si implementano le strutture di controllo in ARM</b>	214		
<b>9 Vettori</b>	215		
<b>10 Etichette e direttive</b>	215		
<b>11 Le eccezioni</b>	216		
<b>12 Interrupt software (SWI)</b>	218		
<b>13 La pipeline e le istruzioni</b>	218		
<b>Approfondimento - Tecnologia</b>			
Formato di una generica istruzione ARM	219		
<b>Laboratorio</b>	222		
<b>ATTIVITÀ 1</b> Utilizzare un emulatore ARM	222		
<b>ATTIVITÀ 2</b> Impariamo a leggere il contenuto della memoria e dei registri della CPU	223		
<b>ATTIVITÀ 3</b> Utilizzare processore ARM su Raspberry Pi	224		
<b>ATTIVITÀ 4</b> Screenshot relativo ai passi descritti nell'ATTIVITÀ 3	225		
<b>ATTIVITÀ 5</b> Utilizzo del Debugger gdb (GNU Debugger)	226		
<b>ATTIVITÀ 6</b> Programmi ARM	228		
<b>Sintesi</b>	233		
<b>Verifica delle conoscenze</b>	234		
<b>Verifica delle abilità</b>	235		
<b>CLIL ARM Architecture</b>	236		
<b>Risorse digitali - Unità 8</b>			
File Aggiuntivi: Codice completo pronto per l'esecuzione	212		
File Aggiuntivi: Codice completo pronto per l'esecuzione	214		
File Aggiuntivi: Codice completo pronto per l'esecuzione	215		

 File Aggiuntivi: Codice completo pronto per l'esecuzione	232	<b>ATTIVITÀ 14</b> Programmare un Single Board Controller (SBC)	281
 Mappa interattiva	233	<b>ATTIVITÀ 15</b> Programmazione dei pin GPIO	283
 Esercizi interattivi	234	<b>ATTIVITÀ 16</b> Output digitale su Raspberry Pi	284
 Esercizi in più	235	<b>ATTIVITÀ 17</b> Input/Output digitale su Raspberry Pi	286
 Audio sintesi in inglese	236	<b>ATTIVITÀ 18</b> Output analogico su Raspberry Pi: PWM	eBook
 Esercizi interattivi in inglese	236	<b>ATTIVITÀ 19</b> Blink di un LED su Raspberry Pi	eBook
<b>UNITÀ 9: Il Physical Computing</b>	<b>237</b>	<b>ATTIVITÀ 20</b> Variazione luce su Raspberry Pi	eBook
<b>1</b> Introduzione	238	<b>ATTIVITÀ 21</b> Variazione della luce di un LED	eBook
<b>2</b> Introduzione all'Internet of Things (IoT)	238	<b>Sintesi</b>	289
<b>3</b> Il controllo delle grandezze fisiche	239	<b>Verifica delle conoscenze</b>	290
<b>Comprendi con l'analogia</b> Controllare un sistema di controllo con retroazione negativa è come salire le scale controllando dove si mettono i piedi	247	<b>Verifica delle abilità</b>	291
<b>4</b> Il Physical Computing con Arduino e Raspberry Pi	250	<b>CLIL Physical Computing</b>	292
<b>Approfondimento - Tecnologia</b>		<b>Risorse digitali - Unità 9</b>	
Il controllo PID (Proporzionale, Integrale, Derivativo)	257	 Attività di laboratorio: Programmare un Single Board Controller (SBC)	279
<b>Approfondimento - Tecnologia</b>		 Approfondimento: Python con Raspberry	283
Sensori e MEMS	258	 Approfondimento: Lavorare con Raspberry	283
<b>Laboratorio</b>		 Attività di laboratorio: Output analogico su Raspberry Pi: PWM	288
<b>ATTIVITÀ 1</b> Controllo ad anello aperto	260	 Attività di laboratorio: Blink di un LED su Raspberry Pi	288
<b>ATTIVITÀ 2</b> Controllo ad anello chiuso	260	 Attività di laboratorio: Variazione luce su Raspberry Pi	288
<b>ATTIVITÀ 3</b> Utilizzo di IDE per lo sviluppo del software	263	 Attività di laboratorio: Variazione della luce di un LED	288
<b>ATTIVITÀ 4</b> Output digitale: lampeggio di un LED	264	 Mappa interattiva	289
<b>ATTIVITÀ 5</b> Input/Output digitale: lettura di un pulsante e lampeggio di un LED	265	 Esercizi interattivi	290
<b>ATTIVITÀ 6</b> Input analogico: lettura di un potenziometro	267	 Esercizi in più	291
<b>ATTIVITÀ 7</b> Output analogico: PWM	269	 Audio sintesi in inglese	292
<b>ATTIVITÀ 8</b> Contatore binario 0-3 con un pulsante, due LED e un buzzer	270	 Esercizi interattivi in inglese	292
<b>ATTIVITÀ 9</b> Potenziometro e fotocellula con dissolvenza	273	<b>Verifica delle competenze</b>	293
<b>ATTIVITÀ 10</b> Accensione di un LED	277	<b>Orientamento al lavoro</b> Sensori	296
<b>ATTIVITÀ 11</b> Automazione garage	277	Sfida finale - ReadyReflex - Riflessi pronti!	298
<b>ATTIVITÀ 12</b> Programmare un Single Board Controller (SBC)	eBook	<b>Lavoriamo con l'Intelligenza Artificiale</b>	
<b>ATTIVITÀ 13</b> Programmare un Single Board Controller (SBC)	279	Interrogiamo l'Intelligenza Artificiale	299

# 3 Architettura di rete: l'accesso alla rete

## UNITÀ 10: Le reti

1 Introduzione	304
2 I sistemi complessi	304
<b>Comprendi con l'analogia</b> Un sistema complesso è come un ponte, che esiste sia grazie alle singole pietre sia grazie agli archi che le connettono	305
3 Teoria delle reti	306
<b>Comprendi con l'analogia</b> La legge di potenza è come la scuola dei paradossi	308
4 Le reti informatiche	309
5 La classificazione delle reti informatiche	310
6 Protocolli di comunicazione	314
<b>Comprendi con l'analogia</b> Anche gli animali hanno bisogno di un protocollo per comunicare	315
7 Standard internazionali	315
8 Architettura di rete	317
<b>Comprendi con l'analogia</b> I livelli di un'architettura di rete sono come gli strati di un panino	318
9 Il modello ISO/OSI e l'architettura TCP/IP	318
10 Funzionamento dell'architettura a livelli	321
<b>Comprendi con l'analogia</b> Il passaggio dei dati dall'alto verso il basso avviene come la conversazione tra due filosofi che non parlano la stessa lingua	322
11 La rete Internet	325
• <b>Sintesi</b>	327
• <b>Verifica delle conoscenze</b>	328
• <b>Verifica delle abilità</b>	329
• <b>CLIL The computer network</b>	330

## Risorse digitali - Unità 10

	Approfondimento: I principali servizi di Internet	325
	Approfondimento: La storia di Internet	325
	Approfondimento: Internet secondo Internet Society	325
	Mappa interattiva	327
	Esercizi interattivi	328
	Esercizi in più	329
	Audio sintesi in inglese	330
	Esercizi interattivi in inglese	330

## UNITÀ 11: Il Livello Fisico

1 Introduzione	332
2 Schema della comunicazione di Shannon	332
3 Definizione e significato di informazione	333
<b>Comprendi con l'analogia</b> L'unità di informazione è come il dischetto numerato estratto nel gioco della tombola	334
4 Il Livello Fisico e i suoi compiti	335
5 La trasmissione del segnale	336
<b>Comprendi con l'analogia</b> Il passaggio dei bit dal computer al mezzo trasmittivo è come un'iniezione di biglie di due colori	336
6 I mezzi trasmittivi	336
<b>Comprendi con l'analogia</b> Gli elettroni nel cavo di rame e i fotoni nella fibra ottica sono come i veicoli motorizzati e le bici nel traffico nell'ora di punta	339
7 I segnali	342
<b>Comprendi con l'analogia</b> La capacità di un canale di trasmissione è come la capacità di una conduttrice dell'acqua	345
8 Tecniche di codifica	345
9 Trasmissione seriale	346
10 Modalità di trasmissione	349
11 Accesso multiplo al canale	349
<b>Comprendi con l'analogia</b> Utilizzare la CDMA è come estrarre l'informazione da un brusio indistinto	351
<b>Approfondimento - Tecnologia</b>	352
La conversione analogico digitale	352
<b>Comprendi con l'analogia</b> La retina è una sorta di fantastico convertitore analogico-digitale naturale	353
<b>Laboratorio</b>	354
<b>ATTIVITÀ 1</b> Applicare lo schema della comunicazione di Shannon al controllo di una diga	354
<b>ATTIVITÀ 2</b> Mezzi fisici di trasmissione disponibili in Packet Tracer	356
<b>ATTIVITÀ 3</b> La gestione Router Switch con Console cable	eBook
<b>ATTIVITÀ 4</b> Montare un cavo UTP per rete Ethernet	357
<b>ATTIVITÀ 5</b> Studiare il segnale su cavi di rame	eBook
• <b>Sintesi</b>	359
• <b>Verifica delle conoscenze</b>	360
• <b>Verifica delle abilità</b>	361
• <b>CLIL The Physical Layer</b>	362

**Risorse digitali - Unità 11**

	Approfondimento: Onde elettromagnetiche	337
	Approfondimento: Onde e moto armonico	342
	Approfondimento: La voce e le armoniche musicali	342
	Approfondimento: Serie di Fourier	342
	Approfondimento: Tecnica del multilivello	345
	Approfondimento: Due esempi di trasmissione seriale	348
	Attività di laboratorio: La gestione Router Switch con Console cable	357
	Attività di laboratorio: Studiare il segnale su cavi di rame	358
	Cablaggio cavo di rete	358
	Come si fa? Assemblaggio di un cavo di rete UTP	358
	Mappa interattiva	359
	Esercizi interattivi	360
	Esercizi in più	361
	Audio sintesi in inglese	362
	Esercizi interattivi in inglese	362

**UNITÀ 12:****Il Livello Collegamento Dati**

<b>1</b>	Introduzione	363
<b>2</b>	Il Livello Data Link e i suoi compiti	364
<b>3</b>	Framing	364
<b>4</b>	Controllo degli errori di trasmissione	366
<b>5</b>	Controllo del flusso	370
	<b>Comprendi con l'analogia</b> I servizi del livello Data Link sono resi possibili da un'organizzazione simile a quella del trasporto delle merci	370
	<b>Comprendi con l'analogia</b> Le tecniche del controllo di flusso sono simili a quelle utilizzate da Ale e Theo per scaricare a mano un camion pieno di sacchi di farina	375
<b>6</b>	Un esempio di protocollo Data Link: HDLC	376
	<b>Laboratorio</b>	378
	<b>ATTIVITÀ 1</b> Progettare il diagramma di sequenza del protocollo di un «Segnalatore ad acqua»	378

<b>ATTIVITÀ 2</b> Analizzare un esempio d'uso della «finestra scorrevole» di trasmissione nel protocollo HDLC	380
---	-----

<b>ATTIVITÀ 3</b> Esaminare casi tipici di funzionamento del protocollo HDLC	381
<b>• Sintesi</b>	383
<b>• Verifica delle conoscenze</b>	384
<b>• Verifica delle abilità</b>	385
<b>• CLIL The Data Link Layer</b>	386

**Risorse digitali - Unità 12**

	Approfondimento: Protocollo XMODEM	377
	Approfondimento: Protocollo BSC	377
	Approfondimento: Protocollo HDLC	377
	Approfondimento: Protocollo PPP	377
	Approfondimento: Controllo del protocollo PPP utilizzato per il collegamento ADSL del router di casa	377
	Connessione a ADSL tramite PPP	377
	Mappa interattiva	383
	Esercizi interattivi	384
	Esercizi in più	385
	Audio sintesi in inglese	386
	Esercizi interattivi in inglese	386

**Verifica delle competenze**

<b>Orientamento al lavoro</b> Protocolli di rete	388
--	-----

<b>Sfida finale</b>	390
---------------------	-----

**Lavoriamo con l'Intelligenza Artificiale**

Il protocollo TCP/IP e la protezione dei dati	392
---	-----

**Educazione civica**

Lo smart working	394
------------------	-----

L'impatto dell'informatica sull'ambiente	396
--	-----

**Indice analitico**

398
-----

*Redattore responsabile:* Stefano Parravicini, Vittoria Varesio

*Redazione e ricerca iconografica:* Andrea De Porti

*Redazione multimediale:* Marco Bosio

*Tecnico responsabile:* Gianni Cupelli

*Progetto grafico:* Matteo Rossi

*Impaginazione e prestampa:* Fotocomposizione Finotello

*Disegni:* Andrea De Porti

*Copertina:* Matteo Rossi

*Ricerca iconografica per la copertina:* Alice Graziotin

*Art Director:* Carla Nadia Maestri

Si ringraziano: Giulio Angiani per il contributo sugli argomenti "introduzione alle reti di computer e ai protocolli di livello fisico"; Francarturo Bertolotti per la collaborazione alla progettazione editoriale e alla revisione del testo, Francesco Guastavino e Domenico La Fauci per le interviste della rubrica "Il mondo del lavoro".

Le traduzioni dall'italiano all'inglese sono a cura di Temitope Ajileye.

*Proprietà letteraria riservata*

© 2024 D Scuola SpA – Milano

DEALINK, DEAFlix sono marchi concessi in licenza da De Agostini SpA.

1<sup>a</sup> edizione: gennaio 2024

*Le fotografie di questo volume sono state fornite da:* Adobe Stock, iStockphoto, New Picture Library, Shutterstock.

*Immagine in copertina:* Shutterstock

L'editore dichiara la propria disponibilità a regolarizzare eventuali omissioni o errori di attribuzione. Nel rispetto del DL 74/92 sulla trasparenza nella pubblicità, le immagini escludono ogni e qualsiasi possibile intenzione o effetto promozionale verso i lettori. Tutti i diritti riservati. Nessuna parte del materiale protetto da questo copyright potrà essere riprodotta in alcuna forma senza l'autorizzazione scritta dell'Editore.

Il software è protetto dalle leggi italiane e internazionali. In base ad esse è quindi vietato decompilare, disassemblare, ricostruire il progetto originario, copiare, manipolare in qualsiasi modo i contenuti di questo software. Analogamente le leggi italiane e internazionali sul diritto d'autore proteggono il contenuto di questo software sia esso testo, suoni e immagini (fisse o in movimento). Ne è quindi espressamente vietata la diffusione, anche parziale, con qualsiasi mezzo. Ogni utilizzo dei contenuti di questo software diverso da quello per uso personale deve essere espressamente autorizzato per iscritto dall'Editore, che non potrà in nessun caso essere ritenuto responsabile per eventuali malfunzionamenti e/o danni di qualunque natura.

Eventuali segnalazioni di errori, refusi, richieste di chiarimento/funzionamento dei supporti multimediali o spiegazioni sulle scelte operate dagli autori e dalla Casa Editrice possono essere inviate all'indirizzo di posta elettronica [info@deascuola.it](mailto:info@deascuola.it).

# Tema

# 1

# Architettura dell'elaboratore

## Unità

- 1 Il sistema di elaborazione e la CPU
- 2 La memoria
- 3 Le periferiche di I/O
- 4 Il software (con DuplOne)
- 5 Evoluzione dei microprocessori

### Prerequisiti

- Sistema di numerazione binario e sistema di numerazione esadecimale
- Elementi base di programmazione
- Strutture di controllo di un linguaggio di programmazione ad alto livello

### Conoscenze

- Il sistema di elaborazione
- Architettura di von Neumann
- Architettura di una generica CPU
- Prestazioni di un sistema a microprocessore
- Capacità della memoria
- La RAM
- Classificazione delle memorie
- Gerarchia di memoria
- Lo stack
- Struttura di un I/O
- Linguaggio assembly: formato delle istruzioni
- Set di istruzioni

### Abilità

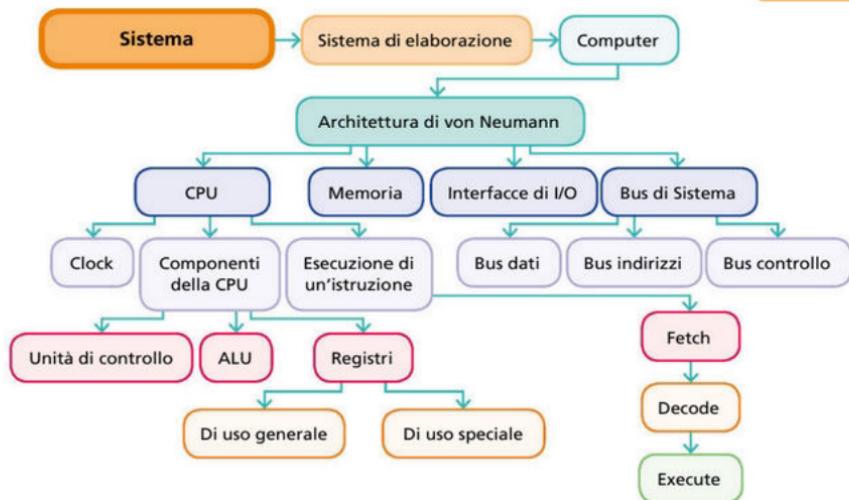
- **Identificare** le principali componenti di un sistema di elaborazione
- **Comprendere** l'architettura di un generico microprocessore
- **Comprendere** come funziona un computer
- **Comprendere** il significato di un programma scritto in linguaggio macchina
- **Valutare** le prestazioni di un sistema a microprocessore
- **Programmare** in linguaggio assembly

### Competenze

- **Configurare, installare e gestire** sistemi di elaborazione dati
- **Scegliere** dispositivi e strumenti in base alle loro caratteristiche funzionali
- **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



# Il sistema di elaborazione e la CPU



## Visione d'insieme

- Architettura di von Neumann.
- Componenti della CPU.
- Esecuzione di un'istruzione.

## 1 Introduzione

A metà del secolo scorso John von Neumann progettò il primo computer moderno e Claude Elwood Shannon introdusse, nella teoria dell'informazione, il concetto di entropia, permettendo di distinguere tra informazione e rumore. Da quel momento il mondo dell'informatica si è sviluppato fino ad arrivare alle ultime applicazioni dell'Intelligenza Artificiale, che inducono a una riflessione su cosa sia l'intelligenza e sulla possibilità che una macchina possa, un giorno, diventare cosciente.

Il problema venne posto già intorno al 1950, quando il matematico Alan Turing suggerì un criterio per stabilire se una macchina è in grado di pensare come un essere umano. Il «test di Turing» consiste nel porre a un soggetto delle domande: se chi pone le domande non è in grado di distinguere le risposte date da una persona da quelle date da una macchina, allora significa che la macchina è intelligente come un essere umano. Ma è davvero così?

Anche un computer che gioca a scacchi appare intelligente, ma quando vince o perde non esulta, né si rattrista; un computer non si stanca di ripetere mille volte la seconda legge di Newton e resta indifferente al fatto che uno studente impari oppure no. Un insegnante in carne e ossa, al contrario, è consapevole di quello che fa: ci mette passione e impegno e soffre se le cose non vanno per il verso giusto. La discussione è tuttavia ben lontana dall'essere chiusa e l'informatica ha ancora molte pagine bianche che aspettano di essere scritte.

L'obiettivo di questa unità è apprendere le conoscenze di base per poter scrivere, un giorno, nuove pagine di questa storia.

### Pit Stop

- Quando è stato progettato il primo computer moderno?
- Che cos'è un sistema?
- Come si identifica lo stato di un sistema?



### Approfondimento

Gli automi

## 2 I sistemi

«Sistema» deriva dal greco *syn* (insieme) e *istemi* (stare), cioè «stare insieme». Ma è sufficiente «stare insieme» come granelli in un mucchio di sabbia o biglie in un sacchetto?

Un **sistema** è un insieme di elementi che interagiscono tra loro al fine di raggiungere un obiettivo comune.

Per esempio: il sistema circolatorio (vasi sanguigni...), il sistema autostradale, un'automobile, un computer.

Spesso gli elementi che costituiscono un sistema sono a loro volta dei sistemi (sottosistemi).

Un sistema è descritto da proprietà che definiscono, istante per istante, lo stato in cui si trova. Per esempio, lo stato di un semaforo è determinato dal colore delle luci.

I sistemi sono classificati in base ad alcune loro caratteristiche, come è riassunto in Tabella 1.

Criteri di classificazione

Natura del sistema	Naturale	Artificiale
	Esistente in natura. <i>Esempio:</i> il corpo umano.	Realizzato dall'uomo. <i>Esempio:</i> il computer.
Tempo	Discreto	Continuo
	Viene preso in considerazione lo stato del sistema in un istante e all'istante successivo. Il sistema «salta» da uno stato all'altro. <i>Esempio:</i> un semaforo passa dal rosso al verde.	Il sistema non ha uno stato ben definito, ma si muove gradualmente da uno stato all'altro. <i>Esempio:</i> la temperatura che varia in una stanza.



<b>Proprietà delle variabili</b>	<b>Dinamico</b> Le variabili di stato cambiano valore nel tempo. <i>Esempio:</i> la pressione atmosferica nel corso della giornata.	<b>Statico</b> Le variabili di stato conservano sempre lo stesso valore. <i>Esempio:</i> un orologio fermo.
<b>Proprietà delle relazioni</b>	<b>Deterministico</b> A una stessa sollecitazione risponde sempre con la stessa risposta. <i>Esempio:</i> un programma per computer che, dato lo stesso input, fornisce sempre lo stesso output.	<b>Probabilistico o stocastico</b> A una stessa sollecitazione può rispondere in diversi modi. <i>Esempio:</i> con una roulette si ottengono risultati apparentemente imprevedibili.
<b>Relazione ingresso/ uscita</b>	<b>Combinatorio</b> L'uscita dipende solo dagli ingressi e non dallo stato interno. I sistemi combinatori sono detti anche «senza memoria». <i>Esempio:</i> il circuito logico NOT.	<b>Sequenziale</b> L'uscita dipende sia dagli ingressi sia dallo stato interno. <i>Esempio:</i> un flip-flop (elemento minimo di memoria)
<b>Interazione</b>	<b>Chiuso</b> Non interagisce con l'ambiente esterno, cioè non riceve nulla dall'esterno e non lascia uscire nulla dall'interno. Sono considerati chiusi anche i sistemi con scambi trascurabili con l'ambiente. <i>Esempio:</i> un barattolo a chiusura ermetica.	<b>Aperto</b> Il sistema interagisce con l'ambiente esterno. <i>Esempio:</i> un sistema biologico o un computer.

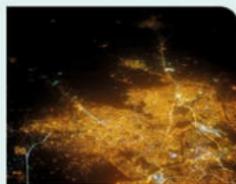
Tab. 1 Classificazione generale dei sistemi.

## Comprendi con l'analogia

In aereo di notte: pensare a un sistema e alle sue parti

Immaginati in viaggio su un aereo, direzione Barcellona. È notte quando inizia la fase di atterraggio. Sei ancora lontano, guardi dal finestrino e la città ti appare come un'enorme «cosa» brillante. Non si distingue nulla, se non un'unica estensione luminosa. A mano a mano che l'aereo si avvicina, cominci a individuare le zone della città, le vie principali; proseguendo e avvicinandoti individui altri dettagli: le auto, i palazzi. Stai entrando nel sistema! Tra poco sarai anche tu un dettaglio di quel sistema.

La città vista dall'alto è un unico sistema, i suoi elementi (autostrade...) sono a loro volta sistemi (sottosistemi).



## 3 Il sistema di elaborazione

Un sistema di elaborazione è un **sistema artificiale aperto**: riceve «qualcosa» dal mondo esterno, lo trasforma e lo restituisce all'esterno modificato. Ciò che entra nel sistema viene chiamato **«input»**, ciò che esce **«output»**. L'operazione di trasformazione dell'input in output viene chiamata **elaborazione** (Figura 1).



### Pit Stop

- Quali caratteristiche ha un sistema di elaborazione?
- Che cosa si intende per «elaborazione»?

Fig. 1 L'elaborazione trasforma dati in input (la materia prima) in output (informazioni).

Il **computer** è un dispositivo costituito da un insieme di elementi in grado di acquisire dall'esterno **dati** e **istruzioni** e produrre in uscita i risultati dell'elaborazione.

L'**hardware** è costituito dalle componenti fisiche (parti elettriche, elettroniche, meccaniche, ...) del sistema.

Il **software** è costituito da sequenze ordinate di istruzioni (i programmi) che indicano al computer come trattare i dati.

### Pit Stop

- 6 Che cosa bisogna fornire a un computer affinché possa risolvere un problema?
- 7 Quali elementi compongono la macchina di von Neumann?

## 4 L'architettura di von Neumann

Per **architettura di un sistema** si intende *come è fatto e come funziona*, quali sono le componenti che lo costituiscono e come interagiscono tra loro.

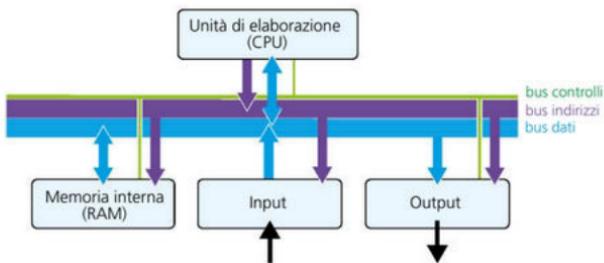
A metà del secolo scorso John von Neumann ideò un sistema per il trattamento delle informazioni in cui l'elaboratore e la memoria sono componenti separate (**Figura 2**).

Nell'architettura di von Neumann per risolvere un problema basta caricare in memoria il relativo programma e i dati su cui eseguire le istruzioni, senza dover modificare l'architettura del sistema. Accade qualcosa di simile quando un cuoco prepara un intero pranzo, dall'antipasto al dolce; il cuoco (elaboratore) non deve far altro che procurarsi gli ingredienti (dati) e la ricetta (programma) di ogni portata (problema da risolvere), ma il suo modo di operare e soprattutto lui stesso sarebbero sempre i medesimi.

Nella memoria viene «inserito» il programma che l'elaboratore deve eseguire: cambiando il programma, cambiano le operazioni svolte.

L'architettura di von Neumann è una delle più utilizzate architetture di un computer. Comprende:

- il **processore o CPU (Central Processing Unit)**. È il cervello del sistema: legge le istruzioni dalla memoria (*fetch*), le interpreta (*decode*) e le esegue (*execute*), una alla volta;
- la **memoria centrale (RAM, Random Access Memory)**. Conserva dati e istruzioni codificati come sequenze finite di cifre binarie 0 e 1 (bit);
- le **interfacce di I/O (Input/Output)**. Collegamenti tra il sistema e il mondo esterno. Servono per connettere al sistema dispositivi utili all'inserimento di dati e programmi (periferiche di input: tastiera, mouse, sensori...) e a fornire all'utente il risultato dell'elaborazione (periferiche di output: monitor, moduli per il collegamento alla rete, attuatori...);
- il **bus di sistema**. È il canale di comunicazione che collega tra loro CPU, I/O e memoria.



→ Fig. 2 La macchina di von Neumann.

## Comprendi con l'analogia

### L'architettura di von Neumann è come le strade di Dodge City

Nel 1878, il «Ford County Globe» pubblica la foto di Dodge City, turbolenta città del Far West americano, che mostra gli edifici che popolano Front Street, l'unica via che l'attraversa. Il saloon è il cuore pulsante della città, attorno al quale gravita la vita degli abitanti e dei forestieri. È il posto in cui si va a bere sregolatamente, ma è anche il luogo in cui si intrecciano affari e relazioni. La banca è lo spazio in cui si conservano i beni. La stazione di cambio delle diligence è la zona che permette la comunicazione con le altre città. Soprattutto c'è Front Street, il vero collante della città, la strada che congiunge gli edifici e permette il passaggio di donne e uomini con le loro merci. Il saloon è come la CPU che è il centro nevrálgico per l'elaborazione dei dati. La banca è la memoria. La stazione di cambio è un sistema di I/O che permette di interfacciarsi con il mondo esterno.

Ogni edificio è raggiungibile dall'unica via che attraversa la città, così come ogni componente della macchina di von Neumann è collegato al bus. Come per entrare e uscire da un edificio si passa da una porta contrassegnata da un'insegna, così per scrivere o leggere un dato in un dispositivo, identificato dal suo indirizzo, si deve abilitare la sua elettronica.

## 5 Il bus di sistema

Esteriormente la CPU è collegata al resto del sistema tramite un insieme di **pin** (piedini), che fanno transitare i diversi segnali dalla CPU verso l'esterno o viceversa, oppure in entrambe le direzioni (bidirezionali). Il numero e il tipo di segnali varia al variare del modello di CPU. L'insieme delle linee costituisce il **bus di sistema**.

Il **bus di sistema** è costituito da linee, collegate alla CPU, su cui viaggiano dei segnali in entrata o in uscita rispetto alla CPU: un segnale per ogni linea. Il numero di linee del bus (ampiezza del bus) varia a seconda dell'architettura. Nel tempo tale numero è aumentato notevolmente, favorendo il miglioramento delle prestazioni del sistema (**Figura 3** a pagina seguente).

Il bus di sistema è costituito da **bus dati** (*data bus*), **bus indirizzi** (*address bus*), **bus controllo** (*control bus*):

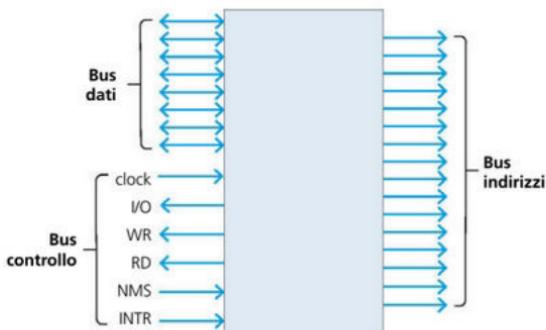
### Pit Stop

- 8 A che cosa serve e da quali parti è costituito il bus di sistema?

	<b>Bus dati</b>	<b>Bus indirizzi</b>	<b>Bus Controllo</b>
<b>Funzione</b>	Trasferimento di dati e istruzioni. Ogni linea trasporta un bit del dato o dell'istruzione.	Indirizzare (selezionare) una cella di memoria o una periferica di I/O.	Consentire alla CPU il controllo e il coordinamento delle altre unità evitando conflitti e collisioni sull'unico bus dati condiviso.
<b>Direzione</b>	Il trasferimento è bidirezionale, avviene in parallelo ed è sempre scandito da un orologio interno al sistema (clock). Si comporta come una strada a senso unico alternato, in cui i dati transitano dalla CPU verso la memoria o le periferiche e viceversa.	Il trasferimento è monodirezionale dalla CPU verso la memoria/periferiche. Si comporta come una strada a senso unico su cui la CPU invia l'indirizzo dell'unità a cui intende mandare/ricevere i dati.	Le linee del bus trasportano ciascuna un segnale al quale è affidato un determinato compito. Ogni segnale può essere in entrata o in uscita, in base alla funzione che deve svolgere. A differenza degli altri bus non può essere letto come un insieme di bit: i suoi bit non fanno parte di un medesimo «messaggio», ma sono tutti indipendenti e con significato proprio.
<b>Dimensione</b>	Il numero di linee del bus dati determina quanti bit di un indirizzo o di un'istruzione possono essere trasferiti contemporaneamente (in un unico ciclo di clock). Un bus dati a 16 linee, per esempio, può trasferire fino a 16 bit per volta. Nei computer attuali il bus dati è a 64 bit o più.	Il numero di linee del bus determina il numero di indirizzi possibili (cioè il numero di celle di memoria che possono essere raggiunte). Per esempio, un bus indirizzi a 16 linee può fisicamente indirizzare fino a $2^{16}$ locazioni di memoria: da 0 a 65535.	Il numero di linee dipende dall'architettura.

**notabene**

Le operazioni di **lettura da memoria** coinvolgono dati o istruzioni, mentre una **scrittura in memoria** è sempre riferita a un dato (risultato dell'esecuzione di un'istruzione).



→ Fig. 3 Collegamento tra CPU e bus di sistema.

L'analogia e gli esempi che seguono illustrano il funzionamento dei bus.

## Comprendi con l'analogia

### Un bus a $n$ bit è come un'autostrada a $n$ corsie

Un trasferimento di  $n$  bit in parallelo, tramite un bus dati a  $n$  linee, può essere pensato come un viaggio di  $n$  auto su un'autostrada a  $n$  corsie. Supponiamo sia  $n = 8$ . All'inizio del viaggio le auto sono disposte ognuna su una corsia; al via (impulso di clock) le auto partono e iniziano il viaggio percorrendo la propria corsia in parallelo con le altre. Al termine si troveranno alla fine della strada (termine del trasferimento). Non è importante che cosa è accaduto durante il viaggio: si considera solo lo stato alla partenza e quello all'arrivo.

## Esempio 1

### Rappresentazione di un indirizzo

Consideriamo un bus indirizzi a 16 linee. In un certo istante la configurazione 0000000001011010 rappresenta l'indirizzo 005A<sub>16</sub>.

Con un bus indirizzi a 16 linee è possibile indirizzare fino a  $2^{16}$  locazioni di memoria utilizzando gli indirizzi compresi tra 0000000000000000<sub>2</sub> e 1111111111111111<sub>2</sub> (65535<sub>10</sub>).

## Esempio 2

### Segnali del bus controllo

Sulle linee del bus controllo (READ, WRITE, I/O, MEM, INT, CLOCK,...) passano i segnali che consentono alla CPU di coordinarsi con le altre unità del sistema ed effettuare determinate operazioni. Un caso tipico è relativo alle operazioni di scrittura/ lettura.

- Il segnale «Write» informa l'unità di destinazione (memoria o periferica) che sul data bus è presente un dato valido per essere memorizzato.
- Il segnale «Read» chiede all'unità di destinazione di immettere sul data bus i dati di cui dispone, per poterli leggere.

Una linea del bus controllo permette alla CPU di selezionare l'unità di memoria o quella di ingresso/uscita. Un caso particolare è quello in cui è la periferica a richiedere l'attenzione della CPU (*Interrupt Request*): in questo caso si tratta di un segnale di ingresso alla CPU che risponde alla periferica attivando un segnale di «richiesta accettata» (*Interrupt Acknowledge*).

## La parola a... John von Neumann

John von Neumann è stato un grande matematico ungherese. Prese parte a diversi progetti che portarono all'architettura di calcolatori che porta il suo nome, e che è alla base degli odierni sistemi di elaborazione.

Nel 1944 lavorò al progetto del calcolatore elettronico ENIAC, che, per la prima volta, realizzava la «macchina universale» che il matematico Alan Turing aveva ideato nel 1936, basandosi sulla logica binaria di George Boole, concepita a metà del XIX secolo.

Nel 1945 von Neumann pubblicò un manoscritto («First draft of a Report on the EDVAC», che puoi reperire in rete) in cui presentava i diversi blocchi logici che compongono il modello di architettura della sua macchina:

- un blocco che contiene organi specializzati, finalizzati all'esecuzione delle operazioni aritmetiche elementari  $+, -, \times, \div$ ;
- un organo di controllo centrale in grado di garantire la corretta sequenza delle operazioni che sono svolte dai diversi blocchi;
- una memoria capace di contenere gli operandi e i risultati delle operazioni aritmetiche;
- una serie di organi atti al trasferimento di informazioni numeriche, o di altro genere, dall'esterno all'interno della macchina, e viceversa.

Vissé negli Stati Uniti fino alla sua morte avvenuta nel 1957.

## 6 Le componenti della CPU

La **CPU** (*Central Processing Unit*) o **processore** è il cuore del sistema: legge dati e istruzioni dalla memoria centrale (RAM) o dalle periferiche di input, esegue le istruzioni e scrive i risultati in memoria o sulle periferiche di output.

Le istruzioni vengono lette ed eseguite una alla volta, nell'ordine con cui sono presenti nella memoria.

La **Figura 4** mostra lo schema dell'architettura interna di una generica CPU. L'architettura di una CPU è costituita di tre elementi principali:

- **CU (Control Unit)** o unità di controllo. Governa e impedisce gli ordini di esecuzione all'unità aritmetico-logica e coordina tutte le operazioni legate all'esecuzione delle istruzioni;
- **Registri**. Sono memorie piccole e veloci;
- **ALU (Arithmetic Logic Unit)** o unità aritmetico-logica. Esegue le operazioni aritmetiche (addizione e sottrazione) e logiche (AND, OR, NOT).

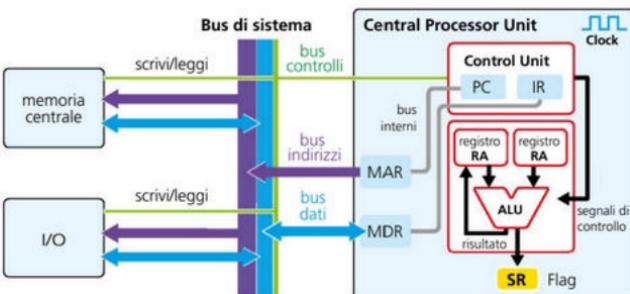


Fig. 4 I componenti di una generica CPU e collegamenti con il sistema.

## 7 L'Unità di Controllo (CU)

L'**Unità di Controllo** ha il compito di coordinare e temporizzare il funzionamento dell'intero sistema. Impartisce gli ordini all'unità aritmetico-logica, alla memoria del computer e ai dispositivi di input e output.

Abilita la scrittura o la lettura da memoria o da I/O, inviando sul bus controllo determinati segnali come, per esempio (Figura 4): RD se lettura, WR se scrittura, I/O se è coinvolta una periferica, MEM se è coinvolta la memoria.

Fanno parte dell'unità di controllo due registri di uso speciale utilizzati per caricare le istruzioni dalla memoria: *Program Counter* e *Instruction Register*.

### notabene

I nomi dei registri utilizzati in questo contesto sono generici e si riferiscono alla loro funzione; in realtà ogni processore è dotato di registri che hanno nomi propri (ad esempio AX, BX, ecc.).

### Pit Stop

- 9 A che cosa servono i registri di uso generale?
- 10 Quali informazioni contiene il PC?
- 11 Quali valori può assumere un flag del SR?
- 12 Quali informazioni contiene il Registro di Stato?

### notabene

Un flag è un bit che segnala se una certa condizione è vera o falsa, in questo caso un errore di calcolo, l'interruzione di programma o semplicemente il riporto di un'operazione aritmetica.

## 8 I registri

I registri sono piccole memorie ad accesso veloce e si dividono in registri **di uso generale** (*general purpose register*) e registri **di uso speciale** (*special purpose register*).

### Registri di uso generale

I registri di uso generale, che in Figura 1 abbiamo chiamato genericamente A e B, sono registri di appoggio usati dalla CPU per memorizzare temporaneamente gli operandi e i dati utilizzati per l'esecuzione di un'istruzione. Il numero e la dimensione dei registri dipendono dalla specifica CPU. Ogni registro è identificato da un nome. Normalmente esiste un registro che è origine e destinazione di molte istruzioni e che prende il nome di **accumulatore**, indicato in genere con la lettera A.

### Registri di uso speciale

- **PC (Program Counter)**. Contiene l'indirizzo di memoria centrale in cui è presente la successiva istruzione che la CPU eseguirà.
- **SR (Status Register, registro di stato)**. Ogni singolo bit di questo registro identifica un «flag» il cui significato è legato allo stato del processore. Ogni flag può assumere solo due valori: «1» logico, quando è ON, oppure «0» logico, quando è OFF.

Alcuni bit sono «bit di stato», altri «di controllo». Il valore dei primi è modificato in seguito al risultato di un'operazione aritmetico-logica. I bit di controllo segnalano invece lo stato in cui può trovarsi la CPU, per esempio quando un programma in esecuzione viene interrotto.

La tabella seguente fornisce alcune indicazioni sui flag più comuni:

Flag	Descrizione	Utilizzo
ZF	Zero Flag	È posto a 1 (settato) se il risultato dell'ultima operazione aritmetico-logica è zero.
CF	Carry Flag	È posto a 1 se il risultato dell'ultima operazione aritmetico-logica ha generato un riporto (o prestito).
SF	Flag di segno	È posto a 1 se il risultato dell'ultima operazione aritmetico-logica è un valore il cui bit più significativo è 1 (se la codifica è in complemento a 2 allora si può dedurre che il numero è negativo).
IF	Interrupt Flag	Se posto a 1 abilita la richiesta di interrupt.

- **SP (Stack Pointer, puntatore allo stack).** Contiene l'indirizzo della cima dello stack. Lo stack è una particolare area di memoria, simile a una pila di piatti, in cui i dati possono essere inseriti ed estratti solo dalla cima.
- **IR (Instruction Register, registro istruzione corrente).** Contiene il codice operativo dell'istruzione da eseguire.
- **MAR (Memory Address Register, registro degli indirizzi della memoria).** Contiene l'indirizzo della memoria principale in cui (o da cui) i dati o le istruzioni devono essere trasferiti.
- **MDR (Memory Data Register, registro dei dati della memoria).** Contiene dati e istruzioni che transitano dalla memoria verso i registri della CPU e i dati che vengono scritti in memoria.

### ■ Accesso ai registri

La **stato della CPU** è rappresentato dai valori presenti in tutti i suoi registri, in un certo istante.

Il trasferimento tra ALU, CU e registri avviene tramite bus interni.

Alcuni registri sono accessibili via software perché il loro contenuto può essere letto o scritto dalle istruzioni di un programma. Altri registri invece non sono accessibili al programmatore. Per esempio, è possibile utilizzare i registri di uso generale (ad esempio il registro A e il registro B) nelle istruzioni di un programma, ma non è possibile usare i registri PC, IR, MAR, MDR che sono registri nascosti e utilizzati dalla CPU come memoria di appoggio per l'esecuzione delle istruzioni.

### Pit Stop

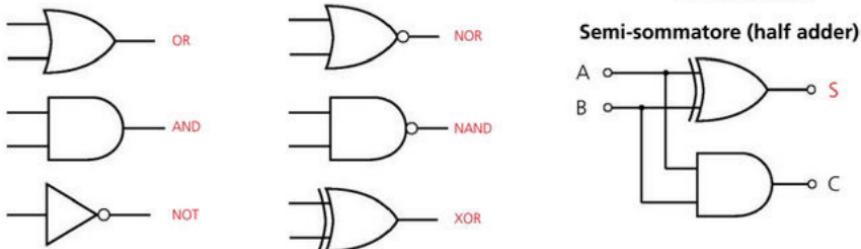
13 Tutti i registri sono accessibili via software?

## 9 L'Unità Aritmetico-Logica (ALU)

L'**Unità Aritmetico-Logica** esegue le operazioni aritmetiche (somma, sottrazione e anche moltiplicazione e divisione, se previste) e logiche (AND, OR, NOT). I dati su cui operare sono presenti nei registri di uso generale e il risultato di un'operazione è posto anch'esso in un registro di uso generale. Dopo un'operazione aritmetico-logica il registro di stato viene aggiornato in base al risultato dell'operazione.

La ALU è realizzata con un insieme di porte logiche opportunamente collegate (*rete combinatoria*), che provvedono a eseguire tutte le operazioni (Figura 5).

↓ Fig. 5 L'ALU contiene delle porte logiche che, unite tra loro, creano circuiti più complessi. Il semi-sommatore, per esempio, esegue la somma di due bit (A e B) utilizzando i circuiti XOR e AND. S è l'uscita corrispondente alla somma, C al riporto.



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Le prime ALU erano in grado di eseguire nativamente solo le operazioni più semplici (addizione, sottrazione, shifting di bit ecc.) e le operazioni logiche booleane (AND, OR, XOR e NOT). Le operazioni più complesse, come le moltiplicazioni, venivano ottenute ripetendo somme in successione.

L'evoluzione dell'elettronica ha reso possibile all'ALU l'esecuzione di operazioni aritmetiche più complesse delle semplici somme e sottrazioni. Tuttavia per i calcoli matematici più sofisticati (come i calcoli in virgola mobile) sono stati spesso utilizzati componenti esterni specializzati (coprocessori). Anche questi, con il passare del tempo, sono stati integrati nella CPU.

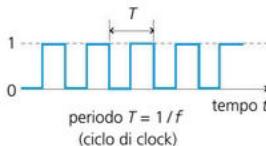
### Pit Stop

- 14 Quali operazioni può eseguire la ALU?
- 15 Che cos'è il clock?
- 16 Che relazione c'è tra la velocità di esecuzione della CPU e la frequenza del segnale di clock?

## 10 Il clock

Il clock è un orologio interno alla CPU. Il segnale generato è periodico ([Figura 6](#)). Tutte le operazioni che la CPU svolge sono scandite dal **ciclo di clock**, che determina anche la velocità con cui si svolgono le operazioni di elaborazione. Maggiore è la frequenza di clock, maggiore è la velocità di lavoro della CPU.

La frequenza di clock  $f$  si misura in Hz e corrisponde al numero di cicli eseguiti in un secondo (periodo). La frequenza  $f$  è l'inverso del periodo  $T$ . Per comodità si usano i multipli degli hertz: kHz, MHz e GHz.



[Fig. 6](#) Il segnale di clock.

### Comprendi con l'analogia

Il clock è come il capovogna che nelle antiche triremi scandiva il ritmo di voga

Le triremi romane disponevano di 180 vogatori che dovevano necessariamente remare in sincronia. Un capovogna scandiva il ritmo con un martello o con un canto a cui si associano i vogatori per poter stare a tempo. Era cruciale per l'equipaggio adattarsi al ritmo di voga, specialmente in battaglia, quando assumeva una frequenza frenetica. Nello storico film *Ben Hur*, del 1959, il cambiamento del ritmo di voga è ben illustrato nella scena della battaglia navale.



### notabene

- 1 kilohertz (kHz)  
= 1000 Hz (mille cicli al secondo)
- 1 megahertz (MHz)  
= 1 000 000 Hz  
(un milione di cicli al secondo)
- 1 gigahertz (GHz)  
= 1 000 000 000 Hz  
(un miliardo di cicli al secondo)

## 11 Esecuzione di un'istruzione

Il processo completo per l'esecuzione di un'istruzione avviene in tre fasi (Figura 7).

### 1. FETCH: prelievo dell'istruzione dalla memoria.

- Il contenuto del Program Counter, che individua l'indirizzo della cella di memoria da cui prelevare l'istruzione, viene copiato nel registro MAR che si interfaccia con il bus indirizzi.
- L'indirizzo contenuto in MAR viene caricato sul bus indirizzi e viene attivata la cella di memoria corrispondente.
- Il codice operativo dell'istruzione, contenuto nella cella di memoria indirizzata, è posto nel bus dati per essere trasferito nel registro MDR.
- MDR contiene il codice dell'istruzione.
- Il contenuto di MDR viene trasferito, tramite il bus interno alla CPU, nel registro IR per essere decodificato.

### 2. DECODE: interpretazione dell'istruzione.

- Il codice dell'istruzione contenuto nel registro IR è decifrato (decodificato) e interpretato. Se la decodifica indica che l'istruzione è più lunga di una localizzazione di memoria, la CPU legge anche il resto dell'istruzione e incrementa il PC del numero di locazioni lette.

### 3. EXECUTE: esecuzione dell'istruzione.

- L'istruzione è eseguita.
- Se l'istruzione prevede la necessità di accedere nuovamente alla memoria per ottenere altri dati, vengono nuovamente coinvolti i registri MAR e MDR.
- L'esecuzione di un'istruzione comporta uno scambio di segnali sulle linee del bus di controllo. Questo scambio avviene sotto la supervisione della Control Unit.

### notabene

Talvolta la fase di decodifica è considerata come la parte iniziale di quella di esecuzione.

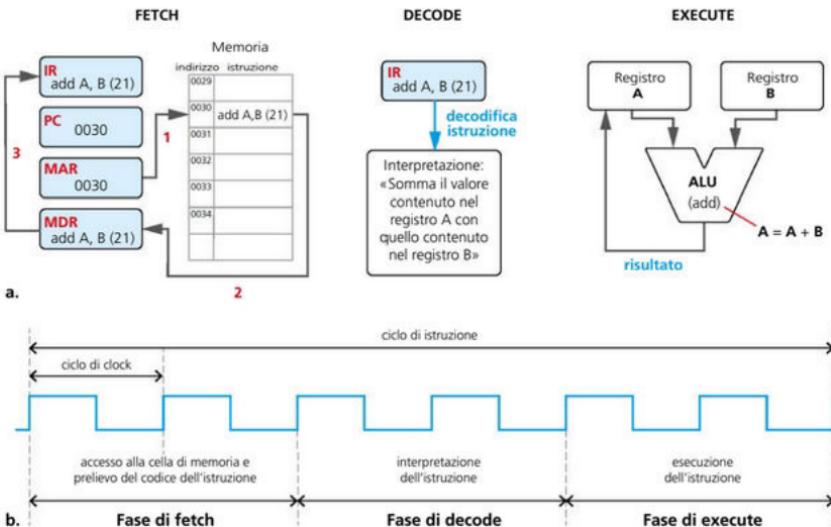
### Pit Stop

- 17 Quali sono le tre fasi di esecuzione di un'istruzione?

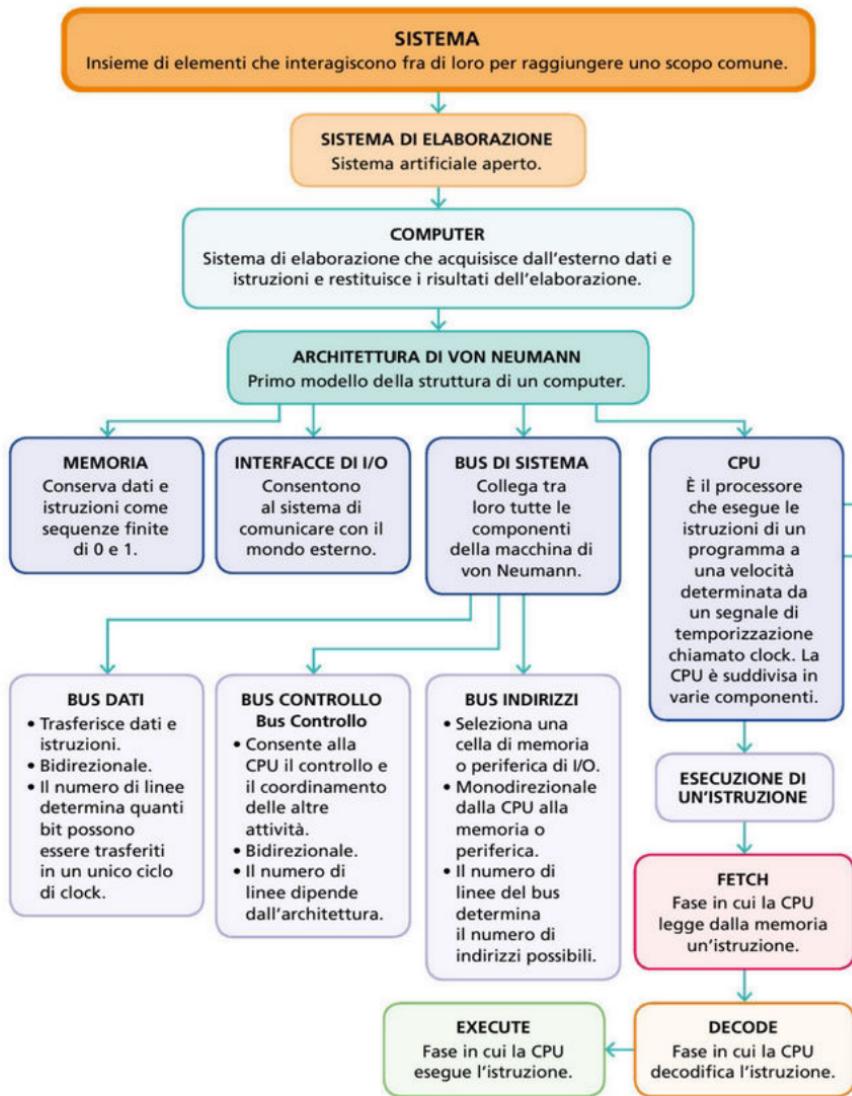
Fig. 7a La fase di fetch prevede:

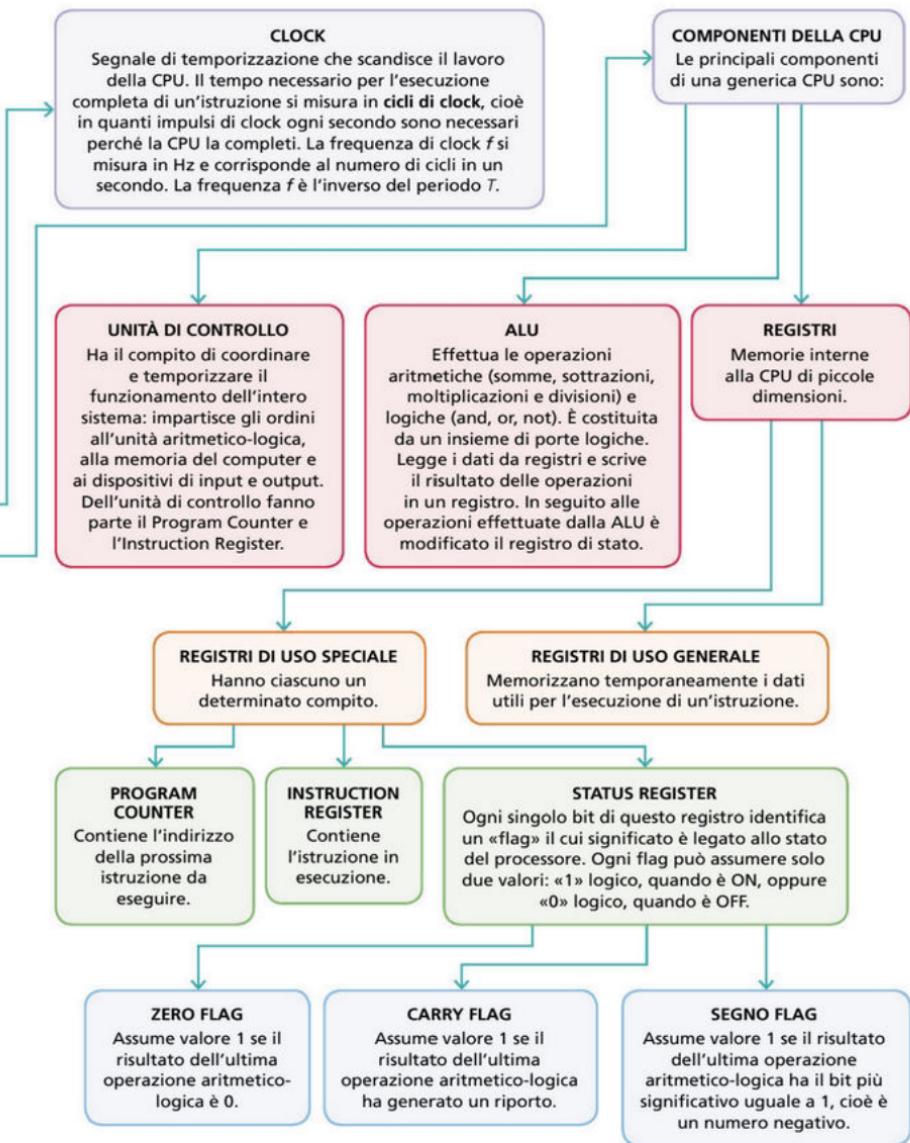
- l'accesso in memoria;
- il prelievo del codice (21 è la codifica esadecimale dell'istruzione add A, B);
- il deposito in IR.

Fig. 7b Andamento qualitativo del ciclo completo di un'istruzione scandita dal clock.



# SINTESI





# Verifica delle CONOSCENZE

Unità 1



## Vero o Falso?

- 1 Il funzionamento della macchina di von Neumann prevede che per ogni istruzione vengano eseguite tre fasi nel seguente ordine: fetch, execute, decode.
- 2 Un computer è un sistema chiuso perché un utente non vede la memoria e il processore.
- 3 Un programma è un insieme di istruzioni.
- 4 Per INPUT si intende tutto ciò che è in ingresso al sistema.
- 5 Gli elementi di un sistema sono sottosistemi.
- 6 Un'istruzione può modificare un dato.
- 7 Un dato può modificare un'istruzione.
- 8 La memoria esegue le istruzioni, il processore le contiene.
- 9 I registri della CPU sono memorie di grandi dimensioni.
- 10 Il registro PC contiene la successiva istruzione da eseguire.
- 11 La configurazione del registro di stato rappresenta lo stato della CPU in un determinato istante.
- 12 La ALU effettua operazioni aritmetiche.
- 13 La CPU legge dalla memoria dati e/o istruzioni.
- 14 La CPU scrive in memoria dati e istruzioni.
- 15 Il bus dati è bidirezionale.
- 16 La CPU è un sistema artificiale.
- 17 I registri della CPU sono sistemi di I/O.
- 18 Il Program Counter è un registro.
- 19 La ALU è esterna alla CPU.
- 20 Nella fase di fetch la CPU preleva un'istruzione dalla memoria.
- 21 Il clock è una unità di misura.

V F

V F

V F

V F

V F

V F

V F

V F

V F

V F

V F

V F

V F

V F

- 22 Il bus controllo si assicura della correttezza delle istruzioni.

V F

- 23 La CPU può effettuare letture dalla memoria.

V F

- 24 La fase di execute è l'ultima istruzione di un programma.

V F

- 25 La velocità di un processore si misura in secondi.

V F



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

- 26 Un sistema è:

- a) un computer
- b) un insieme di elementi autonomi e isolati
- c) un insieme di sottosistemi chiusi
- d) nessuna delle precedenti risposte è corretta

- 27 Un computer è un sistema:

- a) chiuso
- b) naturale
- c) probabilistico
- d) aperto

- 28 Per «architettura di un sistema» si intende:

- a) come è fatto
- b) come funziona
- c) come è stato progettato
- d) come è fatto e come funziona

- 29 L'architettura di von Neumann:

- a) prevede tre componenti: una memoria, un processore e un bus per il collegamento
- b) prevede un'unica unità che contiene le istruzioni e le esegue
- c) è fatta da programmi e dati
- d) Nessuna delle precedenti risposte è corretta

- 30 Le interfacce di Input/Output:

- a) fanno parte del modello di von Neumann
- b) vengono elaborate dal processore
- c) sono in memoria
- d) elaborano i dati

- 31 Quali sono le caratteristiche fondamentali di un sistema e in che cosa si differenzia un sistema da un aggregato qualsiasi?

- 32 Quali sono le caratteristiche dell'architettura della macchina di von Neumann e le fasi che ne caratterizzano il funzionamento?

- 33 Descrivи le caratteristiche principali dei tre bus che costituiscono il bus di sistema.

- 34 Come avviene l'esecuzione di un programma?

- 35 Dove devono essere i dati e le istruzioni che compongono un programma che deve essere eseguito?

- 36 Cosa sono e quali sono i registri di uso speciale?

- 37 Descrivи le principali componenti di una CPU.

- 38 Descrivи in che modo vengono utilizzati i registri SP (Stack Pointer), PC (Program Counter) e il registro di stato.

# Verifica delle ABILITÀ

Unità 1

## Rispondi alle seguenti domande.

- 39** Completa la tabella fornendo due esempi di sistema naturale e sistema artificiale, motivando le risposte.

Sistema naturale	Esempio 1	
	Esempio 2	
Sistema artificiale	Esempio 1	
	Esempio 2	

- 40** Considera il sistema artificiale «caffettiera moka». Individua le componenti principali, gli input e gli output

Componenti:

Input:

Output:



- 41** Scrivi un programma per cucinare gli spaghetti aglio, olio e peperoncino, evidenziando gli ingredienti.

- 42** Descrivi, avvalendoti di un disegno, il modello di von Neumann, mettendo in evidenza le quattro componenti.

- 43** Considera un bus dati a 8 linee. Quanti indirizzi si possono utilizzare? Scrivi in binario e in esadecimale l'indirizzo minimo e massimo.

- 44** Che cosa significa che una CPU lavora a 1 MHz?

- 45** Completa seguendo l'esempio.

1 kHz equivale a 1000 Hz

1 MHz equivale a \_\_\_\_\_

1 GHz equivale a \_\_\_\_\_

- 46** Elenca le caratteristiche di un sistema di elaborazione fin qui studiate che influiscono sulle sue prestazioni.

- 47** Data una CPU con un clock a 2 GHz e che richiede in media 4 cicli per svolgere un'operazione, calcola il tempo necessario per eseguire un programma costituito da 25 000 istruzioni.

- 48** Siano A e B due registri a 8 bit di una generica CPU. Completa la tabella seguente indicando il valore dei bit del registro di stato in base ai valori dei registri e alle operazioni effettuate.

A	B	operazione	ZF	CF	SF
34	45	A + B			
54	54	A - B			
68	124	A - B			
148	32	A - B			
168	182	A + B			
34	74	A and B			



Esercizi in più



## The computing system

### Abstract

A system is a set of autonomous elements, which interact in order to achieve a common goal. The systems are classifiable according to their characteristics.

The computing system is a specific system able to receive inputs from the external world, process (or transform) them, and return the output.

A computer is a computing system.

The computer architecture is based on the von

Neumann model, which includes a processor, the central memory, I/O interfaces, all interacting by the system bus. The «computer's brain» is the Central Processing Unit (CPU), which processes the basic instructions. The CPU is composed by the Arithmetic Logic Unit (ALU), the Control Unit (CU) and the registers. The processor operates at a defined frequency, determined by the clock speed, measured in Hz.

### Questions

#### Answer all questions

- 1 What is a system?
- 2 Which are the components of the von Neumann model?
- 3 Describe the main parts of a CPU.
- 4 What is the role of a Program Counter?

#### Choose the correct answer



- 5 The von Neumann architecture is best defined by which of the following?
  - a Mass storage system
  - b Program output device
  - c Stored-program computer
  - d RAM memory

#### 6 The fetch phase:

- a writes into the RAM
- b reads from the RAM
- c executes an instruction
- d reads from the keyboard

#### 7 The CPU:

- a contains the programs
- b executes the instructions
- c writes the instructions in the RAM
- d reads the instructions from the address bus

#### 8 The program counter:

- a contains the next instruction to execute
- b contains the address of the next instruction to execute
- c contains the current instruction
- d contains the stack top address

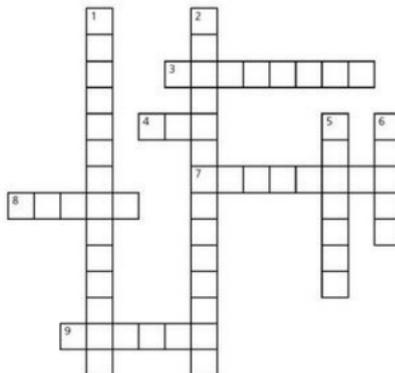
### Crosswords

#### Across

- 3 The physical parts of a computer.
- 4 Executes arithmetic and logical expressions.
- 7 Stores data and instructions.
- 8 Read phase of an instruction.
- 9 The process of interpreting instructions.

#### Down

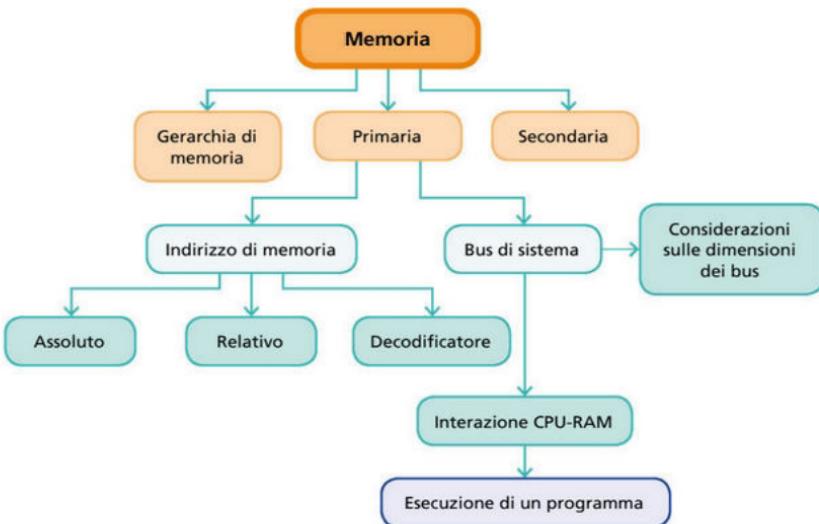
- 1 Contains the address of the next instruction.
- 2 Includes the zero flag and carry flag.
- 5 A communication system that transfers data and instructions within a computer.
- 6 Unit of measurement of CPU speed.





# Unità 2

## La memoria



### Visione d'insieme

- La memoria di un sistema di elaborazione.
- Misura della capacità della memoria.
- Interazione tra la CPU e la RAM: esecuzione di un programma.

## 1 Introduzione: memoria primaria e memorie secondarie

### Pit Stop

- 1 In quale memoria vengono archiviati permanentemente i dati?

La memoria è la parte del sistema di elaborazione che memorizza, in modo permanente o temporaneo, dati e istruzioni. Esistono vari tipi di memorie che si differenziano per tecnologia di costruzione, capacità, tempi di accesso, utilizzo. Dal punto di vista funzionale se ne distinguono due tipi: la memoria centrale o primaria e le memorie secondarie o di massa.

La **memoria primaria o centrale** è realizzata con tecnologia a semiconduttore ed è quella con cui la CPU interagisce direttamente.

Le **memorie secondarie o di massa**, invece, servono per archiviare in modo permanente dati e programmi. Dispongono di una grande capacità di archiviazione, dell'ordine dei terabyte, e sono basate su diverse tecnologie: magnetiche (dischi e nastri), ottiche (DVD) e a stato solido (SSD).

### Comprendi con l'analogia

**La memoria primaria è come la memoria a breve termine del nostro cervello, mentre quella secondaria è come la nostra memoria a lungo termine**

In piedi, dietro la finestra, guardo il traffico sotto di me. Mi ricordo improvvisamente di una mia amica che riconosce le persone dalla targa dell'auto che sta guidando. Sto ancora pensando a questa curiosa associazione quando mi squilla il cellulare. Rispondo e qualcuno mi chiede quanto spendo per la corrente e quanti metri cubi di gas consumo al mese. Involontariamente cerco di fare questi calcoli e poi chiudo la telefonata. Molti ricordi sono contenuti permanentemente in una parte del mio cervello e vengono riportati, a fronte di qualche stimolo sensoriale, in una «memoria di lavoro» che occupa una parte diversa del cervello.

La memoria di lavoro è temporanea e limitata (memoria a breve termine). Conserva piccole quantità di informazioni solo per poche decine di secondi: mentre faccio il calcolo dei metri cubi non posso pensare ad altro e poco dopo me ne dimentico.

La memoria a lungo termine, invece, conserva molti più ricordi e, sebbene più complessa e flessibile, si comporta come la memoria secondaria di un computer.

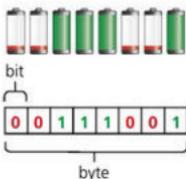


Fig. 1 Un byte è costituito da 8 bit.

Fig. 2 Un nibble è costituito da 4 bit.

Una word è costituita da 16 bit.

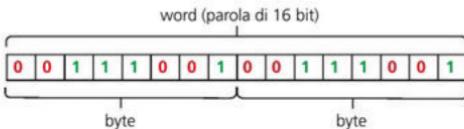
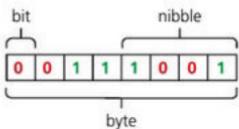
## 2 Capacità della memoria

In informatica tutte le informazioni, numeri, parole, immagini, audio... sono registrate nella forma di sequenze di **bit**. Un bit è una cifra binaria (**binary digit**) che può valere 0 o 1 e codifica lo stato fisico di un segnale. Per esempio, la memorizzazione di un bit 0 può codificare un valore di tensione di 0V, mentre un bit 1 corrisponde a 5V.

Lo scambio di dati e istruzioni, la loro elaborazione e memorizzazione avviene utilizzando sequenze composte da un numero prestabilito di bit.

Una sequenza di 8 bit costituisce un **byte** ed è l'unità minima della memoria (Figura 1).

Altre unità di misura sono il **nibble** (sequenza di 4 bit) e la **word** (sequenza di 16 bit) (Figura 2).



**Pit Stop**

**2 Che cos'è un bit?  
E un byte?**

La capacità di memoria è la quantità di byte che la memoria può contenere. Si misura in byte, o in multipli del byte.

Le unità di misura dei multipli seguono quelle convenzionali (k, M, G, ...), ma si riferiscono a potenze di 2 e non a potenze di 10.

Per esempio, 1 kB non corrisponde a 1000 byte, ma a 1024 (ossia  $2^{10}$ ), che è la potenza di 2 più vicina a 1000.

La **Tabella 1** riporta le tipiche unità di misura della capacità di memoria.

↓ Tab. 1

Unità di misura della capacità della memoria			
Nome	Simbolo	Equivale a	Esempio
bit	-	$2^0$ bit = 1 bit (0/1)	Un pixel in bianco e nero o un flag
byte	-	$2^3$ bit = 8 bit	Un carattere alfanumerico
kilobyte	kB	$2^{10} = 1024 \approx 10^3$ byte	Circa 15 righe di caratteri alfabetici o numerici
megabyte	MB	$2^{20} = 1048576 \approx 10^6$ byte	Un libro di circa 500 pagine
gigabyte	GB	$2^{30} = 1073\,741\,824 \approx 10^9$ byte	Circa 500000 pagine
terabyte	TB	$2^{40} \approx 10^{12}$ byte	Un milione di libri di 500 pagine ciascuno
petabyte	PB	$2^{50} \approx 10^{15}$ byte	180 volte la Biblioteca del Congresso degli Stati Uniti d'America
exabyte	EB	$2^{60} \approx 10^{18}$ byte	180000 volte la Biblioteca del Congresso degli Stati Uniti d'America
zettabyte	ZB	$2^{70} \approx 10^{21}$ byte	180 milioni di volte la Biblioteca del Congresso degli Stati Uniti d'America
yottabyte	YB	$2^{80} = 10^{24}$ byte	180 miliardi di volte la Biblioteca del Congresso degli Stati Uniti d'America

La **Figura 3** mette a confronto i rapporti tra diverse quantità di bit e byte con gli stessi rapporti tra volumi. In questo modo ci possiamo fare meglio un'idea dell'ordine di grandezza associato alle unità di misura.

↓ Fig. 3 Confronto tra diverse capacità.



Qualunque memoria, per quanto «grande», è comunque limitata. Questo ha molte implicazioni dal punto di vista della gestione delle informazioni.

**Pit Stop**

**3** Qual è l'unità di misura della capacità di memoria?

Per esempio:

- la rappresentazione di un numero con la virgola secondo lo standard IEEE 754 comporta delle approssimazioni che possono generare degli errori;
- la codifica di un numero negativo in complemento a 2 è basata sul fatto che la rappresentazione di un numero è su  $n$  bit, con  $n$  finito;
- se un'applicazione software richiede più memoria di quella fisicamente disponibile si ricorre allo stratagemma della «memoria virtuale»: il sistema operativo si serve della memoria secondaria per simulare una memoria centrale più grande.

**Comprendi con l'analogia**

Una memoria finita è come un foglio a quadretti, un quadretto è come un bit

Quando prendiamo appunti per non perdere informazioni, possiamo utilizzare un foglietto, un blocchetto, un quadernino, un quadernone... Qualunque sia il formato scelto del foglio, avremo sempre un numero di quadretti finito.

Ogni quadretto è come un bit!

**3 La memoria primaria o centrale**

La memoria primaria o centrale può essere:

- **RAM (Random Access Memory)**: è una **memoria volatile**, ad accesso diretto, da cui la CPU legge dati e/o istruzioni e su cui la CPU scrive i risultati delle elaborazioni. Random Access Memory letteralmente significa «memoria ad accesso casuale». La parola «casuale» potrebbe trarre in inganno: la traduzione più appropriata è «memoria ad accesso diretto» e significa che il tempo necessario per accedere a una locazione di memoria è indipendente dalla sua posizione;
- **ROM (Read Only Memory)**: è una **memoria permanente di sola lettura** che, per esempio, in un Personal Computer contiene i programmi per l'avvio del sistema.

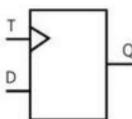


Tabella di verità

D	Q
0	0
1	1

↑ Fig. 4 Flip-Flop di tipo Delay. Il dato (0/1) in ingresso (D) viene trasferito in uscita (Q) in corrispondenza dell'impulso di sincronizzazione (T).

Da un punto di vista circuitale l'elemento minimo della memoria RAM è il **flip flop**, un circuito che può risiedere in due stati stabili (bistabile). In Figura 4 è illustrato il flip-flop di tipo D, che è il circuito su cui si basano le memorie veloci e i registri della CPU.

Il dato (0/1) in ingresso (D) viene trasferito in uscita (Q) in corrispondenza dell'impulso di sincronizzazione (T).

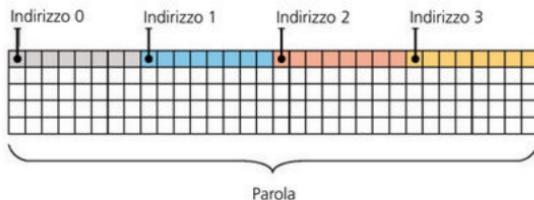
L'uscita si mantiene in questo stato (dato memorizzato) fino a una nuova variazione del segnale in T, anche se l'ingresso D cambia.

La situazione è simile a quando stiamo vedendo un film e chiudiamo gli occhi. La retina trasmette al cervello l'ultima scena che abbiamo visto, che rimane in memoria anche se la trama del film prosegue con altri frame. Per memorizzare una nuova scena dobbiamo riaprire gli occhi. Le scene del film sono i dati in ingresso che possono variare in continuazione. La chiusura degli occhi corrisponde al segnale di sincronizzazione, che memorizza il dato in ingresso e lo mantiene in uscita.

## 4 | L'indirizzo di memoria

La memoria centrale è suddivisa in celle o locazioni che possono essere identificate tramite il loro **indirizzo**, cioè la posizione della cella nella memoria. Tutte le celle hanno la stessa dimensione. La memoria è poi organizzata in parole (word). Ogni parola è formata da 1 o più byte. L'indirizzo di memoria è associato a una unità indirizzabile che in genere è il singolo byte, mentre la parola è riferita all'organizzazione della memoria e quindi alla dimensione dei dati.

Per esempio, una parola potrebbe essere costituita da 4 byte, ciascuno con un indirizzo. La prima locazione di memoria ha indirizzo 0. La **Figura 5** mostra un esempio di organizzazione della memoria in parole di 4 byte in cui l'unità minima indirizzabile è il singolo byte. La dimensione delle parole e delle unità minime indirizzabili dipendono dall'architettura. In genere le parole sono di 8, 16, 32, 64 bit, e sono trasferite alla CPU tramite il bus dati.



### Pit Stop

- 5 Che cosa è un indirizzo di memoria?

◀ **Fig. 5** Le celle di memoria organizzate in «parole». La prima riga evidenzia la suddivisione di una parola in 4 byte, l'unità minima indirizzabile.

### Comprendi con l'analogia

La memoria è come una cassetiera; le locazioni di memoria sono come i cassetti; l'indirizzo di una locazione è come il numero di un cassetto.

La memoria può essere paragonata a una cassetiera: un cassetto è una locazione, referenziata tramite un'etichetta o tramite la sua posizione. La cassetiera serve per contenere qualcosa.

La cassetiera in figura è costituita da cassetti numerati. Il numero identifica la posizione di un cassetto nell'intera cassetiera e serve per identificare il cassetto. Numero di cassetto e contenuto del cassetto sono, evidentemente, diversi: il numero identifica la posizione del cassetto, mentre il contenuto è ciò che è riposto nel cassetto. Potrà sembrare banale, ma molti errori di programmazione sono dovuti a questa confusione di concetti.



Per **indirizzo assoluto** (o *fisico*) si intende la posizione di una locazione di memoria a partire da quella di indirizzo 0.

Per **indirizzo relativo** (o *logico*) si intende la posizione di una locazione di memoria rispetto a un'altra. Per esprimere un indirizzo relativo occorrono due numeri: l'indirizzo della locazione di riferimento (**base**) e la distanza da essa (**spiazzamento, offset**).

La **Figura 6** mette in evidenza una locazione di memoria di indirizzo relativo (2, 3): la locazione di indirizzo assoluto 5 ha indirizzo relativo (2, 3) perché occupa la posizione 3 a partire dalla 2. Possiamo dire che ha distanza 3 dalla 2. Ogni locazione ha un unico indirizzo assoluto a cui possono corrispondere più indirizzi relativi.

0	01101101
1	01000000
2	...
3	
4	
5	
6	

↑ **Fig. 6** Locazioni di memoria e indirizzi associati. È evidenziata in arancione la locazione di indirizzo assoluto 5 e relativo di base 2 e offset 3 (2, 3).

## Comprendi con l'analogia

L'indirizzo di una locazione di memoria è come la numerazione della pagina di un libro

La situazione è analoga alla numerazione delle pagine di un libro. Possono essere numerate dalla prima all'ultima con un unico numero (assoluto), come nella figura di sinistra, oppure possono avere una numerazione che associa a ogni pagina due numeri: numero di capitolo e numero di pagina all'interno di quella sezione (numero relativo al capitolo), come nella figura di destra.

come il problema

unco per i byte di  
tto, la CPU è in  
uesto implica che

171

alla 0 alla 15) servono  
di input/output. Le  
cioè 65 536.

ato dalla RAM, è pos-  
ferimento; in questo

3.4

## 5 La decodifica degli indirizzi

### Pit Stop

- Come fa la CPU ad accedere a una cella di memoria RAM?

Un circuito integrato (*chip*) di memoria contiene, oltre alle singole celle, dei circuiti di decodifica, il cui scopo è riconoscere l'indirizzo per poter accedere, in lettura o scrittura, alle celle di memoria.

Le locazioni sono disposte in una matrice (quadrata o rettangolare) strutturata in righe e colonne, in cui ogni locazione occupa una posizione identificata dal numero di riga e dal numero di colonna.

Per attivare la cella di memoria è necessario un **decodificatore**, un circuito logico il cui scopo è riconoscere l'indirizzo per poter accedere, in lettura o scrittura, alle diverse caselle disposte nella matrice.

Un decodificatore è un circuito logico con  $n$  ingressi e  $2^n$  uscite in cui, per ogni configurazione di ingresso, una sola uscita è attiva.

Lo schema di principio è mostrato nella Figura 7. A ogni incrocio riga-colonna della matrice è presente un elemento di memoria (concretamente un flip-flop) che viene selezionato come si fa nella «battaglia navale».

L'indirizzo di  $n$  bit, che è presente nel MAR e viene posto sull'*address bus*, è ripartito in righe ( $m$ ) e colonne ( $k$ ) ( $n = m + k$ ).

Un «decodificatore di riga» seleziona la riga e un «decodificatore di colonna» seleziona la colonna. La presenza contemporanea dell'indirizzo sull'*address bus* e dei segnali di abilitazione del chip (pin CS – *Chip Select*) e del segnale di *Read/Write* (pin R/W) del control bus, permettono di abilitare la cella in lettura o scrittura.

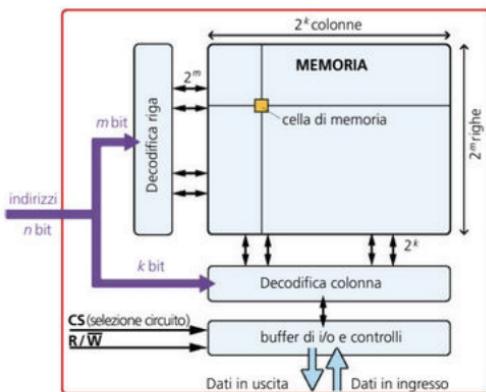


Fig. 7 Schema di funzionamento di un decodificatore di indirizzi.

### notabene

Per le operazioni di Read e Write possono essere utilizzate due linee distinte, oppure, come nel caso di

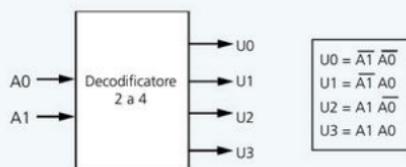
Figura 7, un'unica linea. In questo caso il segnale R/D è attivo alto (1), mentre W/R è attivo basso (0) (nello schema è contrassegnato con  $\overline{W}$ ).

## Esempio 1

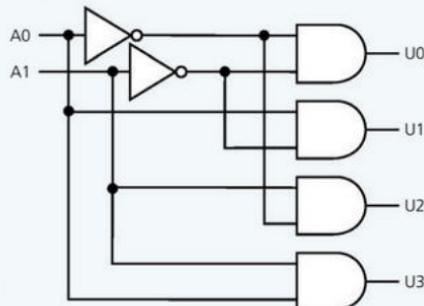
Un decodificatore è un circuito logico con  $n$  ingressi e  $2^n$  uscite in cui, per ogni configurazione di ingresso, una sola uscita è attiva. Per esempio, la **Tabella 2** mostra la tabella di verità di un decodificatore con 2 ingressi e 4 uscite di cui una sola è attiva (1) mentre le altre sono disattive (0) (**Figure 8 e 9**).

INGRESSI		USCITE			
A0	A1	U0	U1	U2	U3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

↑ Tab. 2



↑ Fig. 8 Simbolo del decodificatore con la funzione booleana corrispondente alla tabella di verità.



→ Fig. 9 Schema elettrico del decodificatore 2 a 4.

## Esempio 2

Consideriamo un sistema con bus dati a 8 bit e bus indirizzi a 16 bit con 4 chip di memoria da 16 kbyte per un totale di 64 kbyte ( $2^{16}$ ).

Lo schema nella **Figura 10** di pagina seguente mostra la parte di decodifica degli indirizzi. Per semplicità è rappresentato solo l'address bus che, con le due linee A14 e A15 che entrano nel decodificatore 2 a 4, indirizza uno dei 4 chip di memoria, mentre con i rimanenti 14 bit individua la cella di memoria nel chip.

Supponiamo che l'indirizzo sull'address bus sia:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	0	0	0	1	0	1	0	0	0	0	1	1	0	1	1
Chip di memoria	Colonna										Riga				



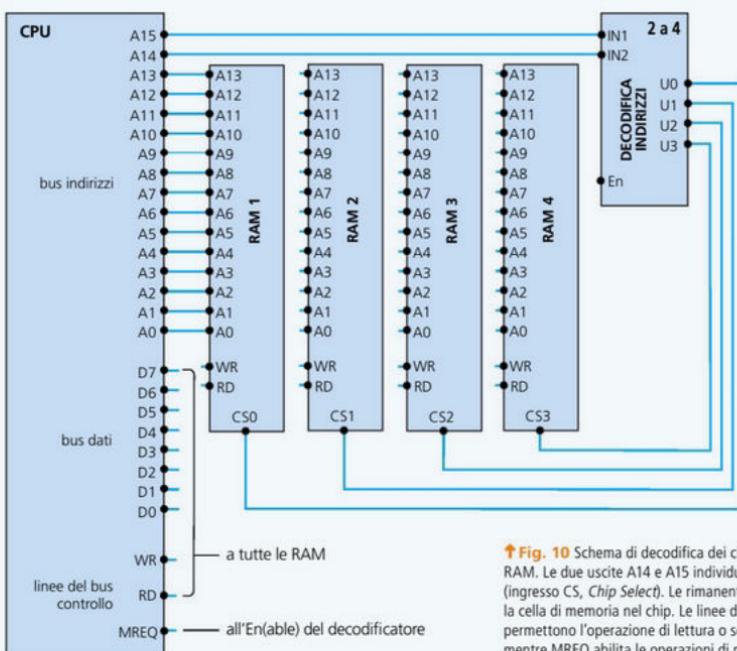


Fig. 10 Schema di decodifica dei chip di RAM. Le due uscite A14 e A15 individuano il chip (ingresso CS, Chip Select). Le rimanenti indirizzano la cella di memoria nel chip. Le linee di RD e WR permettono l'operazione di lettura o scrittura, mentre MREQ abilita le operazioni di memoria.

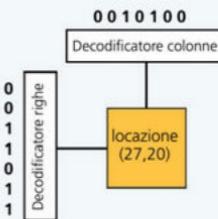
I bit A15 e A14 sono utilizzati per la selezione del chip attraverso un decodificatore esterno alla memoria con 2 ingressi e 4 uscite ( $2 \times 4$ ). I restanti bit sono utilizzati per identificare una cella di memoria su quel chip.  
Supponiamo di usare

- i bit più significativi A13 - A7 per le colonne;
- i bit meno significativi A6 - A0 per le righe.

Il decodificatore di colonna attiverà la colonna 20 (20 decimale corrisponde a  $0010100_2$  binario).

Il decodificatore di riga attiverà la riga 27 (27 decimale corrisponde a  $0011011_2$  binario).

Sarà quindi attivata la locazione (27,20) del chip 2.



### notabene

Nel sistema possono esserci più chip di memoria, pertanto l'indirizzo di memoria deve identificare in modo gerarchico, tramite un decodificatore esterno, prima il chip (bit più significativi) e poi la locazione di memoria su quel chip.

## 6 Il bus di sistema e l'interazione tra CPU e RAM

L'esecuzione di un'istruzione necessita di procedure che coinvolgono l'unica via di comunicazione tra le unità del sistema: il bus.

Il **bus dati** è bidirezionale e permette il trasferimento di informazioni tra CPU e RAM (o periferiche di I/O) e viceversa. La CPU controlla queste due modalità di transito tramite le linee del *control bus* (**bus controllo**). Le linee del bus controllo trasportano ciascuna un segnale preciso, al quale è affidato un determinato compito. Ogni segnale può essere in entrata o in uscita, in base alla funzione che deve svolgere. A differenza degli altri bus, il bus controllo non può essere letto come configurazione di bit; i suoi bit non fanno parte di un medesimo «messaggio», ma sono tutti indipendenti e con significato proprio. La **Tabella 3** riassume il significato di alcuni segnali del control bus.

### notabene

È da precisare che lo stato di attivazione dei diversi segnali potrebbe essere lo stato logico «1» (vero o alto), oppure «0» (falso o basso). In quest'ultimo caso il simbolo con cui è contrassegnato il segnale è caratterizzato dal segno di negazione come nell'algebra booleana. Per esempio il segnale A, se negato, diventa  $\bar{A}$ .

Linea	Funzione	Direzione rispetto alla CPU
<b>Lettura</b> (RD - <i>Read</i> , o OE - <i>Out Enable</i> )	È attivato quando la CPU deve fare un'operazione di lettura da memoria o da periferica; indica al dispositivo sorgente di inviare un nuovo dato sul bus dati	Uscita
<b>Scrittura</b> (WR - <i>Write</i> o STB - <i>Strobe</i> )	È attivato quando la CPU deve fare un'operazione di scrittura su memoria o periferica. Indica che sul bus dati è disponibile un dato valido	Uscita
<b>Memoria</b> (MEM o MREQ - <i>Memory Request</i> )	È attivato quando la CPU deve effettuare un trasferimento (lettura o scrittura) che coinvolge una locazione di memoria	Uscita
<b>Input/Output</b> (I/O o IOREQ - <i>I/O Request</i> )	È attivato quando la CPU deve effettuare un trasferimento (lettura o scrittura) che coinvolge una periferica	Uscita
<b>Clock</b>	È il segnale di temporizzazione	Ingresso
<b>Interrupt Request</b> (INTR o INT)	È il segnale di richiesta interruzione mascherabile da parte di una periferica	Ingresso
<b>Interrupt Acknowledge</b> (INTA)	È attivato dalla CPU quando viene accettata la richiesta di interruzione	Uscita
<b>Interrupt Non Mascherabile</b> (NMI)	È il segnale di richiesta interruzione non mascherabile. Quando la CPU riceve questo segnale deve sospendere il processo in esecuzione	Ingresso

La comunicazione tra CPU e memoria avviene nel modo seguente:

- la CPU, assumendo il ruolo di «master» invia gli ordini di scrittura/lettura a memorie o periferiche;
- la memoria (o la periferica) assume il ruolo di «slave»: si comporta da «sorgente» quando invia i dati sul *data bus* o da «destinazione» quando si predispone a riceverli;
- il bus dati trasporta le informazioni scambiate;
- il bus indirizzi individua l'unità sorgente/destinazione;
- per distinguere se l'istruzione è di lettura o di scrittura, viene attivato uno dei segnali **Read** o **Write**;
- per distinguere se l'indirizzo che viene posto sull'address bus si riferisce a una memoria o a una periferica viene attivato uno dei segnali **MREQ** o **IOREQ**.

↑ **Tab. 3** Il nome dei segnali e il loro stato, con il segnale che si attiva se il livello fisico è «alto» o «basso», varia in base all'architettura della CPU.

La Figura 11 mostra la temporizzazione dei segnali di controllo (RD, WR, MREQ), degli indirizzi e dei dati per i cicli di lettura e scrittura che coinvolgono la memoria. Il segnale di clock impone il sincronismo alle operazioni che, di norma, si sviluppano su diversi periodi.

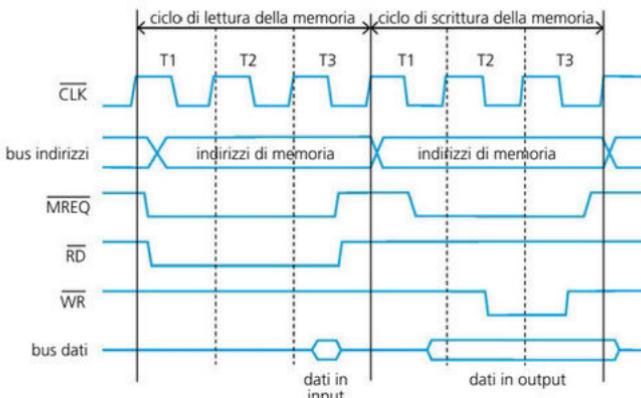


Fig. 11 Una generica lettura dalla memoria prevede l'attivazione della linea RD (attiva bassa). La scrittura di un dato in memoria si serve del segnale WR (attivo basso). In entrambi i casi il segnale MREQ (attivo basso) diventa attivo quando il valore dell'indirizzo della cella di memoria, presente sul bus indirizzi, è stabile.

## 7 Ciclo di lettura e scrittura

### Ciclo di lettura dalla memoria

Un generico ciclo di lettura dalla memoria è svolto nel modo seguente (Figura 11):

- l'indirizzo della locazione di memoria è posto sull'address bus all'inizio del primo ciclo di clock;
- il segnale MREQ diventa attivo basso quando il bus indirizzi è stabile (durante il ciclo T2). Il chip di memoria viene abilitato;
- la linea di RD diventa attiva per indicare che i dati letti dalla memoria sono posti sul bus dati («dati in input»). Il segnale di WR rimane inattivo (stato alto) per tutto il tempo dell'operazione di lettura.

### Ciclo di scrittura nella memoria

Un generico ciclo di scrittura in memoria è svolto nel modo seguente:

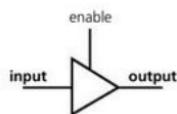
- l'indirizzo della locazione di memoria è posto sull'address bus all'inizio del primo ciclo di clock;
- il segnale MREQ diventa attivo basso quando il bus di indirizzi è stabile; il chip di memoria viene abilitato;
- la linea di WR diventa attiva bassa quando i dati sul data bus sono stabili («dati in output»). Il segnale di RD rimane inattivo (alto) per tutto il tempo dell'operazione di scrittura.

Il bus dati dev'essere assegnato a un dispositivo alla volta per evitare conflitti. Per questo è fondamentale che i dispositivi si possano disconnettere dal bus

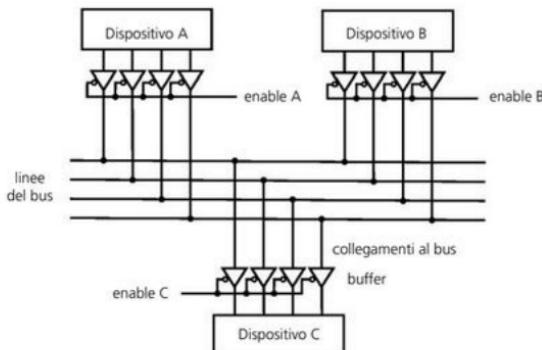
**Pit Stop**

- 7 Quali segnali dei bus sono coinvolti nelle operazioni di lettura e scrittura?

quando è necessario. A questo proposito la porta logica dello stadio di uscita di un dispositivo che si collega al bus (*buffer*), aggiunge ai due stati classici (0 e 1) un terzo stato (***three-state*** o ***tri-state***) che rende l'uscita un circuito aperto e permette all'unità di «staccarsi» dal bus. In realtà l'uscita del buffer assume una impedenza molto elevata tale da comportarsi come un circuito aperto (**Figura 12**). Questo significa che si possono collegare due o più buffer allo stesso filo senza interferenze con altri buffer tri-state (**Figura 13**).

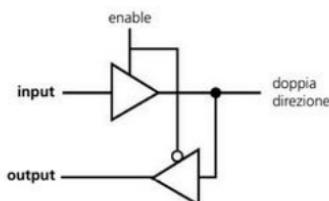


Enable	Input	Output	Stato
0	0	alta impedenza	input → enable → output
0	1	alta impedenza	input → enable → output
1	0	0	input → enable → output
1	1	1	input → enable → output



◀ **Fig. 12** Il buffer *three-state* (o *tri-state*) si comporta come un interruttore aperto (stato di «alta impedenza») quando il pin di *Enable* è posto a 0. Altrimenti si comporta come un buffer convenzionale.

La bidirezionalità del bus dati si ottiene accoppiando due buffer, come illustrato nella **Figura 14**, abilitando separatamente i due buffer. Consente di selezionare l'invio o la ricezione dei dati sul bus da parte della RAM o di periferiche. Il buffer bidirezionale controlla il traffico del segnale per prevenire conflitti sul bus assicurando che almeno uno dei buffer sia in stato di alta impedenza, consentendo all'altro lato di accedere ai fili del bus: il buffer inferiore è soggetto a un segnale di abilitazione opposto a quello superiore, assicurando che il bus possa essere solo in lettura o in scrittura.



◀ **Fig. 14** Il buffer bidirezionale.

◀ **Fig. 13** I dispositivi sono abilitati separatamente (linea di *Enable*) in modo da evitare contese sul bus.

## 8 Considerazioni sulla dimensione dei bus

### ■ Dimensione del bus indirizzi

Il numero di linee del bus indirizzi determina il numero di indirizzi possibili e quindi la quantità di memoria utilizzabile. Ogni linea del bus indirizzi trasporta un bit a cui è associato il valore logico 0 o 1. La configurazione degli  $n$  bit rappresenta l'intero indirizzo: un bus a  $n$  linee può indirizzare fino a  $2^n$  locazioni di memoria.

### Esempio 3

#### Spazio di indirizzamento

- Con **8 linee del bus indirizzi** si indirizzano:  $2^8 = 256$  locazioni di memoria. Gli indirizzi vanno da 0 a 255, cioè  $2^8 - 1$  (in binario, da 0000000 a 1111111; in esadecimale, da 0h a FFh).
- Con **16 linee del bus indirizzi** si indirizzano:  $2^{16} = 65\,536$  locazioni di memoria, cioè 64 KB.
- Con **20 linee del bus indirizzi** si indirizzano:  $2^{20} = 1\,048\,576$  locazioni di memoria, cioè 1 MB.
- Con **32 linee del bus indirizzi** si indirizzano:  $2^{32} = 4\,294\,967\,296$  locazioni di memoria, cioè 4 GB.

### ■ Dimensione del bus dati

La dimensione della locazione di memoria e dei registri della CPU è legata alla dimensione del bus dati che trasferisce ogni bit su una sua linea.

La **Tavola 4** mostra un esempio della rappresentazione di un sistema con bus dati a 8 bit e bus indirizzi a 16 bit. È posta in evidenza la distinzione tra indirizzo di una locazione di memoria e suo contenuto, espressi sia in binario sia in esadecimale.

↓ Tab. 4 Rappresentazione di un sistema con bus dati a 8 bit e bus indirizzi a 16 bit. Indirizzi e contenuto della memoria sono espressi in binario o in esadecimale.

Indirizzo della locazione di memoria		Contenuto della locazione di memoria		
decimale	esadecimale	binario	binario	esadecimale
0	0	0000000000000000	11001101	CD
1	1	0000000000000001	11001001	C9
2	2	0000000000000010	00110110	36
3	3	0000000000000011	00000000	00
4	4	0000000000000100	01000010	42
5	5	0000000000000101	00010100	14
6	6	0000000000000110	00100010	22
7	7	0000000000000111	00100111	27
...	...	...	...	
66	42	000000001000010	11110100	F4
...	...	...		
65535	FFFF	1111111111111111	111111100	FC

## 9 Esecuzione di un programma

Per comprendere meglio l'interazione tra CPU e RAM utilizziamo l'esempio seguente che mostra come avviene l'esecuzione di un programma secondo l'architettura di von Neumann.

In questo contesto useremo il linguaggio di programmazione Assembly che è un linguaggio di basso livello. Ogni architettura ha le sue specifiche istruzioni Assembly.

Consideriamo, per semplicità, una CPU con bus indirizzi a 16 linee e bus dati a 8 linee. Ipotizziamo che le locazioni di memoria e i registri di uso generale siano di 1 byte.

Supponiamo che la CPU debba eseguire l'istruzione scritta in linguaggio Assembly di un generico microprocessore:

```
mov R1,[0042h]
```

che significa: «Trasferisci il contenuto della locazione di memoria di indirizzo 0042h nel registro R1». Supponiamo inoltre che l'istruzione sia codificata in binario su 3 byte:

00110110	36h	codice operativo: indica che cosa deve fare l'istruzione
00000000	00h	indirizzo di memoria da cui prelevare il dato
01000010	42h	da trasferire

Questo significa che il programmatore scrive `mov R1, [0042h]` e l'assemblatore, un apposito software che descriveremo più avanti, traduce l'istruzione in linguaggio binario:

```
00110110 00000000 01000010
```

Supponiamo che nella locazione 0042h sia contenuto il valore F4h ( $11110100_2$ ). Al termine dell'esecuzione dell'istruzione nel registro R1 sarà presente il valore F4h.

## ■ Stato iniziale della memoria

Indirizzi		Contenuto RAM
hex	bin	
0000	0000000000000000	
0001	0000000000000001	
0002	0000000000000010	00110110
0003	0000000000000011	00000000
0004	0000000000000100	01000010
0005	0000000000000101	00010100
0006	0000000000000110	00100010
	...	...
0042h	000000001000010	11110100

} Istruzione da eseguire  
 } Prossima istruzione da eseguire  
 } Dato da trasferire

## ■ Stato iniziale della CPU

Il Program Counter (PC) punta (cioè indirizza) alla locazione 0002h (evidenziata in rosso).

### notabene

Il linguaggio Assembly verrà approfondito in seguito. Qui è introdotto per evidenziare il legame tra CPU, RAM, bus e istruzioni.

### notabene

Specifichiamo che in questo caso il codice operativo è puramente di fantasia, ma ogni macchina ha il suo set di codici ammessi, le istruzioni che è in grado di interpretare.

### notabene

Ricordiamo che il Program Counter è un registro di uso speciale che contiene l'indirizzo della locazione di memoria della prossima istruzione da eseguire.

## 10 Esecuzione dell'istruzione

Riprendiamo l'esempio e vediamo come avviene l'esecuzione dell'istruzione

```
mov R1, [0042h].
```

### ■ Fase di FETCH

- La CPU pone sul bus indirizzi l'indirizzo 0002h.
- Viene attivata la locazione di indirizzo 0002h.
- Il contenuto della locazione 0002h viene posto sul bus dati.
- Il codice operativo dell'istruzione è posto nel registro IR (*Instruction Register*).

### ■ Fase di DECODE

- L'unità di controllo decodifica il codice operativo dell'istruzione, cioè ne interpreta il significato: la CPU dovrà trasferire in R1 il dato posto nella locazione il cui indirizzo è nei byte che seguono il codice operativo (indirizzi 3 e 4 di memoria).
- La CPU preleva, tramite il bus dati, l'indirizzo del dato (i due byte rimanenti) e lo pone nel registro MAR.
- Il contenuto del PC viene modificato e aumentato della lunghezza dell'istruzione appena letta; in questo caso l'istruzione è lunga 3 byte, quindi PC = 0002h + 3, cioè PC = 0005h. In questo modo sarà pronto per puntare alla prossima istruzione da eseguire.

### Pit Stop

- 8 Quali sono le fasi in cui è divisa un'istruzione?
- 9 Dove viene posto il codice operativo di un'istruzione durante la fase di fetch?
- 10 Che ruolo ha il *Program Counter* nell'esecuzione di un programma?

### ■ Fase di EXECUTE

L'esecuzione dell'istruzione consiste nel trasferimento del dato.

- Viene posto sul bus indirizzi il contenuto di MAR (0042h).
- Viene attivata la cella 0042h.
- Tramite il bus dati viene prelevato il contenuto della locazione 0042h (F4h) e trasferito nel registro R1.

A questo punto **il ciclo riprende** con la FETCH dell'istruzione posta all'indirizzo 0005h (prossima istruzione).

## 11 Lo stack

Lo stack è una parte della memoria centrale che può contenere solo dati. È utilizzato in situazioni particolari ed è gestito dinamicamente con politica LIFO (*Last In First Out*): l'ultimo dato scritto nello stack sarà il primo a essere letto, come in una pila di libri o di piatti l'ultimo oggetto collocato è il primo a venire prelevato. L'indirizzo di memoria della cima dello stack è mantenuto nel registro SP (*Stack Pointer*) un registro di uso speciale della CPU (**Figura 15**).

- Per la scrittura: la CPU decrementa SP e, successivamente, trasferisce il dato nella locazione di memoria il cui indirizzo è in SP;

- per la lettura: la CPU trasferisce il dato dalla locazione di memoria il cui indirizzo è in SP e, successivamente, incrementa SP.

Relativamente allo stack questo significa che, posto l'inizio dello stack a un indirizzo, quando è necessario utilizzare locazioni di memoria, vengono usate quelle contigue con indirizzo più basso.

...	
...	
...	
FFFCh	
FFFDh	
FFFEh	
FFFFh	

a.

...	
FFFCh	
FFFDh	
FFFEh	FFEh
FFFFh	FFFe

b.

Prossima locazione utilizzata

Base dello stack

Base dello stack

### notabene

Possiamo pensare che lo Stack Pointer «tiene il segno» per sapere fino a dove è utilizzato lo stack.

◀ Fig. 15 a. Stack vuoto. La prima locazione è all'indirizzo fffffh.  
b. Inserimento di un dato nello stack: viene utilizzata la locazione di indirizzo FFFF – 1 = FFFE.

## Comprendi con l'analogia

### Lo stack è come una pila di libri

Scrivere sullo stack e leggere dallo stack è come mettere e togliere un libro da una pila di libri: si può riporre un libro in cima alla pila (scrittura sullo stack) e prelevare il libro depositato per ultimo (lettura dello stack).



## 12 Operazioni sullo stack

Le istruzioni per scrivere e leggere dallo stack sono (Figura 16):

- PUSH:** prima scrive un dato nello stack all'indirizzo indicato da SP, successivamente il contenuto del registro SP viene decrementato; *PUSH* significa «inserire in cima»;
  - POP:** prima incrementa il contenuto del registro SP e successivamente legge il dato contenuto nella locazione di memoria puntata da SP; *POP* significa «togliere dalla cima».
- L'operazione di POP non cancella il dato prelevato ma, modificando SP, rende disponibile lo spazio da esso occupato.

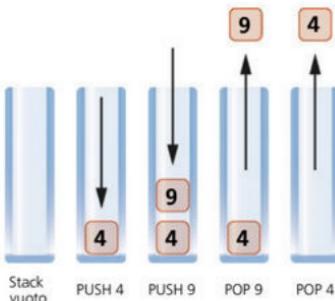
→ Fig. 16 Logica di funzionamento della politica LIFO.

### notabene

Se vengono eseguite di seguito le istruzioni PUSH e POP la situazione dello stack rimane invariata, perché PUSH inserisce un dato e POP lo preleva.

### Pit Stop

- 11 Che cos'è lo stack?
- 12 Cosa si intende per LIFO?
- 13 Quali sono le istruzioni che operano sullo stack?



Lo stack è utilizzato nei linguaggi ad alto livello per allocare le variabili locali (e altri dati correlati) di un metodo o funzione. In questo caso, infatti, anche in presenza di chiamate di funzione nidificate, le ultime variabili allocate sono le prime a essere deallocate. Lo stack è anche usato per salvare lo stato di un processo al verificarsi di un *interrupt*.

## Esempio 4

### Operazioni sullo stack

Consideriamo la situazione di partenza mostrata in Figura 17, in cui la prima locazione utilizzabile è all'indirizzo FF03h (cima dello stack). Supponiamo di avere a disposizione un registro della CPU, R1, che contiene il dato 2:  $R1 = 2$ . La cima dello stack è alla locazione FF03; la prima locazione utilizzabile è all'indirizzo FF03.

La Figura 18 rappresenta la situazione dopo l'operazione PUSH R1 (che significa genericamente: «carica nello stack il contenuto di R1»). Prima viene trasferito il contenuto di R1 nella cima dello stack, poi viene decrementato SP. La successiva locazione utilizzabile è la FF02h.

La Figura 19 mostra la situazione dopo l'operazione POP R1 (che significa genericamente: «preleva un dato dallo stack e mettilo in R1»). Prima viene decrementato il valore SP; poi viene «prelevato» il contenuto della locazione puntata da SP e trasferito in R1. La successiva locazione utilizzabile è la FF03h.

Osservando le figure è possibile notare che il contenuto della memoria non è stato modificato; infatti l'istruzione POP preleva il dato, ma non lo cancella dallo stack. Quello che cambia in realtà è il valore di SP. Dopo la POP,  $SP = FF03h$ , e questo significa che la successiva PUSH utilizzerà quella locazione, eventualmente sovrascrivendo il dato. È lo stack pointer che indica quale locazione è possibile utilizzare!

...	
FF03h	
FF04h	•
...	•
FFF Eh	•
FFFFh	•

SP = FF03h →  
Base dello stack

...	
FF02h	
FF03h	2
FF04h	•
...	•
FFF Eh	•
FFFFh	•

sale  
↑ SP = FF02h →  
Base dello stack

...	
FF02h	
FF03h	2
FF04h	•
...	•
FFF Eh	•
FFFFh	•

↓ SP = FF03h →  
scende  
Base dello stack

† Fig. 17

† Fig. 18

† Fig. 19

## Esempio 5

### Utilizzo dello stack

Siano R1 = 3 e R2 = 5.

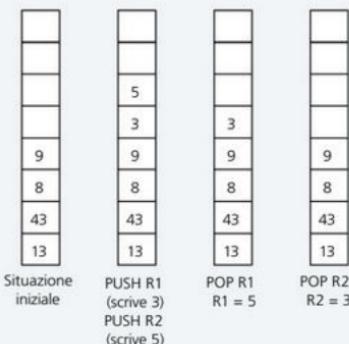
Ci chiediamo che cosa contengono R1 e R2 dopo l'esecuzione delle seguenti istruzioni:

```
PUSH R1
PUSH R2
POP R1
POP R2
```

La situazione è rappresentata nella figura a fianco.

Il risultato è che è stato scambiato il contenuto dei registri R1 e R2.

Occorre fare bene attenzione all'utilizzo delle istruzioni PUSH e POP: un uso errato potrebbe causare notevoli danni!



## 13 Le memorie a semiconduttore

La memoria centrale di un sistema di elaborazione è realizzata con la tecnologia dei semiconduttori. Tale tecnologia nel corso degli anni ha consentito il miglioramento della capacità, dei tempi di accesso e dei costi per *unità di bit* delle memorie.

Le memorie a semiconduttore sono di due tipi (**Figura 20**):

- le **memorie volatili**: conservano i dati finché il chip è alimentato. Le **memorie volatili a semiconduttore** costituiscono la memoria centrale. Rientra in questa categoria la memoria RAM, che può essere di due tipi: DRAM o SRAM;
- le **memorie non volatili o permanenti**: conservano i dati anche in assenza di alimentazione elettrica.

Le memorie non volatili a semiconduttore sono utilizzate per integrare nell'hardware di un dispositivo i microprogrammi (firmware) e i programmi di sistema, come il BIOS, il software necessario all'avvio dei computer.

Sono di questo tipo la memoria ROM (che può essere PROM, EPROM, e EEPROM) e la memoria flash.



### notabene

L'utilizzo di tale tecnologia non si limita alla sola memoria centrale, ma, tramite l'applicazione della tecnologia flash, anche alla memoria di massa per archiviare i dati.



### Approfondimento

I semiconduttori



### Videointervista

I semiconduttori

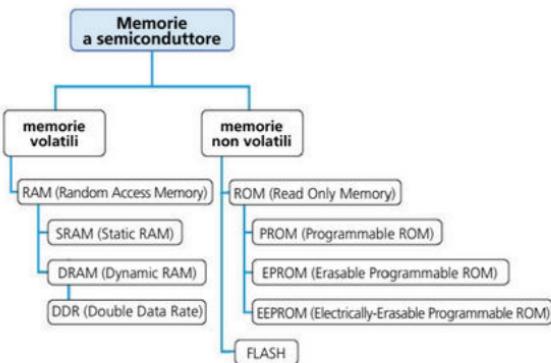


Fig. 20 Memorie a semiconduttore, volatili e non volatili o permanenti.

## 14 La RAM

La **RAM** (*Random Access Memory*) è una memoria volatile che può essere letta e scritta. I dati e i programmi sono mantenuti in memoria fino a quando il sistema è alimentato, in caso contrario si perdono.

*Random Access Memory* letteralmente significa «memoria ad accesso casuale». La traduzione più appropriata è «**memoria ad accesso diretto**»; infatti ogni cella della memoria RAM è indirizzata direttamente, indipendentemente dalla sua posizione. Ciò significa che la CPU impiega lo stesso tempo (dell'ordine dei nanosecondi) per accedere a una locazione, a prescindere dal suo indirizzo. L'accesso diretto si contrappone, in vari ambiti, all'«accesso sequenziale» dove, per reperire un elemento in un insieme ordinato, occorre «passare» da tutti quelli che lo precedono, con tempi di accesso che dipendono dalla posizione dell'elemento. Un accesso diretto è possibile solo se è nota la posizione dell'elemento a cui accedere.

### Comprendi con l'analogia

L'accesso diretto e l'accesso sequenziale possono essere paragonati a due modi diversi di utilizzare il telecomando della TV

I due diversi modi di accesso al contenuto di una memoria, diretto e sequenziale, possono essere paragonati ai due diversi modi di accesso a un canale televisivo concessi dalla maggior parte dei telecomandi.

Se dobbiamo vedere il nostro film preferito e sappiamo su quale canale sarà trasmesso, digitiamo direttamente sulla tastiera del telecomando il numero del canale (accesso diretto).

Se invece non sappiamo su quale canale sarà trasmesso, dobbiamo cercarlo: partiamo dal canale 1 e con l'apposito tasto andiamo al canale successivo finché non troviamo il film che cerchiamo (accesso sequenziale).

Nel primo caso il tempo necessario per andare al canale desiderato prescinde dal canale. Nel secondo caso invece, ci vorrà più tempo ad arrivare al canale scelto.



### Pit Stop

- 14 Che cosa vuol dire «memoria volatile»?
- 15 Che cosa vuol dire «memoria permanente»?
- 16 Che cosa si intende per «memoria ad accesso diretto»?

### notabene

Possedere una memoria di lavoro di parecchi gigabyte (GB) è come possedere un comodo tavolo in cucina su cui lavorare a proprio agio senza dover spostare, ogni volta, le stoviglie nell'armadio per andare a riprenderle poco dopo.

### notabene

Il transistor è un dispositivo utilizzato come interruttore nei circuiti digitali e amplificatore di segnale nei circuiti analogici.

La RAM costituisce la memoria centrale, che nell'architettura di von Neumann rappresenta la **memoria di lavoro**. La sua dimensione incide in modo determinante sulle prestazioni di un qualsiasi sistema di elaborazione: nessun processore, per quanto potente, può fare a meno di una quantità di RAM adeguata. In funzione della tecnologia utilizzata le memorie RAM possono essere di due tipi: **dinamiche (DRAM)** o **statiche (SRAM)**.

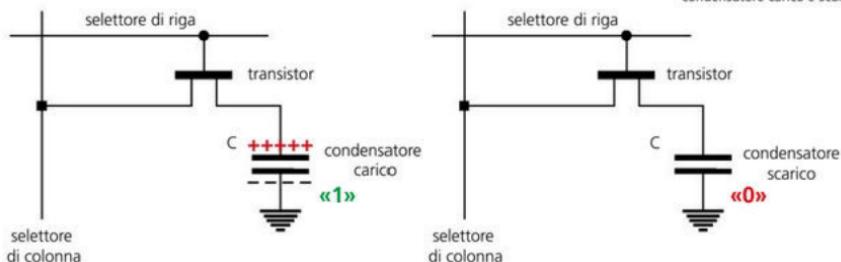
## ■ La DRAM

La **DRAM** (*Dynamic Random Access Memory*) è la RAM dinamica.

Ogni cella di memoria è composta da un *transistor* e da un condensatore in cui l'informazione è memorizzata come carica elettrica (Figura 21). Un condensatore può essere paragonato a un secchio che invece di contenere acqua è riempito di elettroni. Il condensatore carico ha valore logico «1»; se è scarico ha valore logico «0».

Il problema è che il condensatore tende a scaricarsi, come un secchio bucato che va continuamente rabboccato. Per risolvere questo problema la CPU, o il circuito di controllo, devono «rinfrescare» periodicamente il contenuto del condensatore con un'operazione di *refresh* che deve avvenire entro un tempo massimo di qualche millisecondo per mantenere la carica. Il segnale di refresh è un segnale del bus di controllo in uscita dalla CPU.

Fig. 21 Stati logici «1» e «0» in una cella DRAM, corrispondenti al condensatore carico e scarico.



Una cella di memoria dinamica occupa una piccola area del semiconduttore, permettendo una maggiore densità di integrazione, cioè il numero di porte logiche presenti sul chip, a costi ridotti. Il basso costo è il motivo per cui questo tipo di memoria è largamente utilizzata come **memoria primaria** nella maggioranza degli elaboratori.

Una DRAM realizzata con tecnologia **DDR (Double Data Rate)**, «trasferimento dati a doppia velocità», sfrutta simultaneamente due canali per raddoppiare la velocità di trasferimento. Per esempio, una **DDR4** con una frequenza operativa del bus a 1066 MHz può trasferire dati a 2132 MHz.

Le DRAM sono organizzate in banchi chiamati **DIMM (Dual In-line Memory Module)** che si inseriscono nella scheda madre dei computer (Figura 22). Questi banchi ospitano i *chip* di **SDRAM (Synchronous Dynamic Random Access Memory)**, una memoria in cui le operazioni di scambio dati con la CPU sono sincronizzate da un clock esterno.

La velocità dei banchi di RAM presenti in un PC è limitata dalle impostazioni di fabbrica della scheda madre. Ciò serve a ridurre i consumi ed evitare surriscaldamenti. Per aumentare le prestazioni è possibile operare l'*overclocking* (cioè incrementare la frequenza di lavoro), intervenendo sui parametri del BIOS.

Fig. 22 Le DIMM sono banchi di memoria SDRAM che si inseriscono nell'apposito alloggiamento presente nella scheda madre del computer.



Le memorie DRAM per i sistemi mobili e IoT (*Internet of Things*) hanno la necessità di consumare poca energia e di essere efficienti. Le memorie **LPDDR SDRAM (Low-Power Double Data Rate, Synchronous Dynamic Random-Access Memory)** sono progettate apposta per queste esigenze.

## ■ La SRAM

La **SRAM (Static Random Access Memory)** è la RAM statica. È costituita da un flip-flop e non ha bisogno di alcun *refresh* (Figura 23 a pagina seguente). Ciò consente velocità significativamente maggiori di quelle delle DRAM. Tuttavia

**Pit Stop**

- 17 Quali sono le differenze tra DRAM e SRAM?

L'implementazione di una cella di memoria statica richiede molto più spazio di una DRAM e la rende più costosa. La SRAM si utilizza in memorie che hanno bisogno di un'alta velocità e di bassa capacità, come le memorie *cache*.

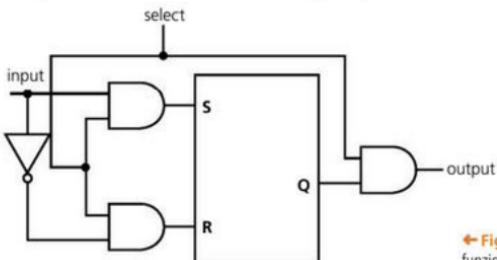


Fig. 23 Schema di funzionamento della SRAM.

## 15 La cache memory

La cache memory è una memoria che logicamente si trova tra la CPU e la memoria centrale DRAM. È una SRAM, quindi più costosa e più veloce della DRAM. È relativamente piccola ed è usata per mantenere più vicine alla CPU le informazioni (dati e istruzioni) statisticamente più richieste, migliorando le prestazioni del sistema. Ci sono vari livelli di memoria cache: livello 1 (L1, integrata nel processore), livello 2 (L2) e livello 3 (L3).



### Approfondimento Metodi di gestione della cache memory

#### Il funzionamento logico della cache memory

La memoria cache è organizzata in blocchi chiamati linee. Le informazioni che la CPU legge dalla memoria centrale durante l'esecuzione di un programma, sono copiate anche nella cache, in modo tale che siano disponibili più rapidamente all'accesso successivo.

#### L'utilizzo della memoria cache per operazioni di lettura

Quando la CPU deve effettuare una lettura, le informazioni (dati o istruzioni) vengono cercate prima nella memoria cache; se sono presenti (*cache hit*) non serve accedere alla memoria RAM, altrimenti (*cache miss*) viene effettuata una lettura nella memoria RAM e le informazioni prelevate vengono memorizzate anche nella cache per i prossimi accessi.

#### L'utilizzo della memoria cache per operazioni di scrittura

Quando la CPU deve effettuare una scrittura, la memoria cache può operare in due modalità:

- **modalità write-through:** la scrittura viene effettuata sia nella cache sia nella memoria centrale (solo l'inizio è simultaneo, perché la scrittura nella cache è più veloce). Questo metodo è semplice da realizzare e non richiede che le linee della cache vengano riscritte nella memoria quando vengono sostituite;
- **modalità write-back:** la CPU scrive solo nella cache. La copia nella memoria centrale è fatta quando la linea di cache è sostituita. In questo modo la scrittura è più veloce. Un miglioramento consiste nell'utilizzare un bit (*dirty-bit*) per indicare se le informazioni nella linea sono state modificate e quindi se devono essere veramente scritte in memoria quando la linea viene sostituita; questo accorgimento permette di non perdere tempo per copiare informazioni che non sono state modificate.

**notabene**

La cache è più piccola della DRAM e le informazioni che contiene sono una copia di una parte della DRAM. Cache e DRAM devono sempre essere allineate.

**Pit Stop**

- 18 A che cosa serve la cache memory?  
19 Che cosa si intende per «cache hit»? E per «cache miss»?

## 16 La ROM

La **ROM** (*Read Only Memory*) è una memoria ad accesso diretto non volatile che può essere solo letta. È scritta una sola volta quando è prodotta e non è modificabile.

Le memorie ROM sono utilizzate per contenere microprogrammi o programmi di sistema. Per esempio, in un PC contengono il software di partenza della macchina (ROM-BIOS).

Le celle di memoria hanno una organizzazione interna simile a quella mostrata nella **Figura 24**. Se all'incrocio della riga e della colonna c'è un transistor, il valore logico conservato è «1», altrimenti è «0».

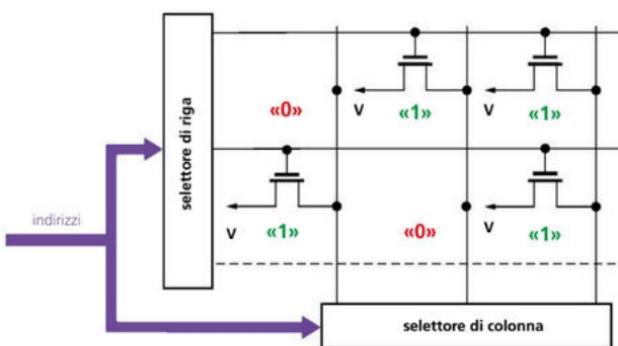


Fig. 24 Il chip è organizzato sotto forma di matrice, in cui ogni cella può contenere un bit di informazione.

### Evoluzione delle ROM

Le ROM sono utilizzate per cablare i programmi direttamente nel silicio (firmware). A partire dalla ROM di base, in cui il codice viene scritto una volta per sempre nel silicio, l'evoluzione della tecnologia ha portato a diverse soluzioni che prevedono la possibilità di riscrivere una ROM in modi e tempi diversi:

- **PROM (Programmable Read Only Memory)**: vengono scritte elettricamente tramite apparecchiature speciali;
- **EPROM (Erasable Programmable Read Only Memory)**: sono più flessibili delle PROM perché, oltre a essere scritte, possono essere cancellate e riscritte. La carica dei transistor nelle celle viene azzerata da raggi ultravioletti che passano attraverso una piccola finestra in quarzo trasparente presente sul chip.
- **EEPROM (Electrically Erasable Programmable Read Only Memory)**: sono più versatili delle EPROM perché possono essere scritte e cancellate elettricamente senza staccare il chip dal circuito stampato che lo ospita. Sono più lente e complesse delle EPROM, perché richiedono tensioni diverse per la lettura e la scrittura;
- **flash**: sono più veloci e flessibili delle EEPROM, perché utilizzano lo stesso valore di tensione per scrivere, leggere e cancellare interi blocchi di memoria con una sola operazione. Sono anche meno complesse e consumano meno energia. Inoltre utilizzano celle multilivello che permettono di registrare il valore di più bit con un solo transistor.

Gruppi di chip di memoria flash costituiscono la base per i sistemi di archiviazione a stato solido (**SSD: Solid-State Drive**) in sostituzione dei dischi a rotazione.

Sono presenti nei dispositivi mobili come smartphone e tablet e nelle chiavette USB. Quando sono utilizzate al posto delle ROM in sola lettura, prendono il nome di flash ROM.

## 17 Confronto tra le memorie a semiconduttore

Possiamo confrontare le caratteristiche delle memorie a semiconduttore esprimendo un giudizio **qualitativo** sui tempi di accesso, le capacità, il numero massimo di operazioni possibili e i costi, come è mostrato nella **Tabella 5**.

Nella tabella:

- il simbolo  $\infty$  (infinito) significa «senza un limite pratico»;
- i simboli +, - esprimono la qualità di una tecnologia rispetto all'altra.

↓ Tab. 5 Comparazione

qualitativa delle memorie volatili e non volatili.

Memoria						
	Volatile		Non volatile			
	SRAM	DRAM	ROM	EPROM	EEPROM	FLASH
alimentazione (necessaria per conservare i dati in memoria)	tensione	tensione con refresh			non necessaria	
numero di operazioni di lettura	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
numero di riscrittture (ordine di grandezza)	$\infty$	$\infty$	0 volte	100 volte	Da 100 000 a 1 milione di volte	Da 10 000 a 1 000 000 di volte
tempo di lettura	+++	++	+	+	+	+
tempo di scrittura	+++	++	non scrivibile	-	-	-
costo per bit	-	+	++	-	-	+++
capacità	+	+++	++	-	-	+++

## 18 Le memorie di massa

I dispositivi che consentono l'archiviazione dei dati (*data storage*) permettono di salvare dati e programmi su dispositivi che li conservano anche in caso di mancanza di alimentazione (memoria non volatile o permanente). Poiché la capacità di queste memorie è decisamente grande, vengono chiamate memorie di massa. Da un punto di vista tecnologico i dispositivi di storage si possono classificare in funzione della tecnologia utilizzata (Tabella 6).

**notabene**

L'attuale esigenza di archiviare enormi quantità di dati ha due aspetti predominanti. Da una parte la necessità di soddisfare le esigenze legate ai big data e all'Intelligenza Artificiale, dall'altro la protezione di perdite di dati accidentali o dovuti ad attacchi e frodi informatiche. In questi casi lo storage dei dati è impiegato anche come sistema di backup.

<b>Supporti magnetici</b>	<b>Disco rigido (Hard Disk Drive – HDD)</b>	È costituito da uno o più piatti, rivestiti di materiale ferromagnetico, che ruotano a una velocità che può superare i 15 000 giri al minuto. Due testine per ogni disco permettono di accedere direttamente ai dati che possono essere letti o scritti (Figura 25). I dischi sono caratterizzati da: capacità di memoria (GB e TB), tempo di accesso e velocità di trasferimento.
	<b>Nastro magnetico (Tape Drive)</b>	Utilizzati soprattutto nel passato per memorizzare grandi quantità di dati. Attualmente si utilizzano ancora nei sistemi di backup. Le unità a nastro si servono di una testina magnetica di lettura/scrivitura per accedere ai dati in modo sequenziale, con conseguente lentezza. La capacità di archiviazione di un nastro arriva a molti TB.
<b>Unità ottiche (Optical Drive)</b>		Utilizzano raggi laser per leggere o scrivere su dischi CD (Compact Disc), DVD (Digital Versatile Disc) e BD (Blu-ray Disc) che sono utilizzati per l'ascolto di musica o la visione di un film o per effettuare backup di dati.
<b>Unità a stato solido (Solid State Drive – SSD)</b>		Sono memorie a stato solido (senza parti in movimento) basate su tecnologia flash, che rende questi dispositivi molto più veloci degli Hard Disk. La capacità di archiviazione può arrivare a diversi TB. Gli SSD non hanno parti in movimento e sono, perciò, più affidabili, efficienti e meno dispendiosi a livello energetico dei dischi rigidi (Figura 26).

↳ Tab. 6



↑ Fig. 25 Hard disk drive.



↑ Fig. 26 SSD interno con i chip di memoria flash.



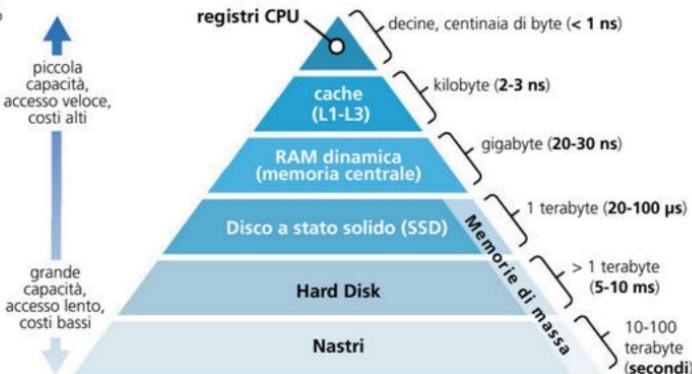
↳ Fig. 27 Un tecnico si accinge a inserire un modulo SAN in un rack.

## 19 La gerarchia delle memorie

Le prestazioni delle memorie si caratterizzano per due parametri: la capacità e la velocità di accesso da parte della CPU. La Figura 28 mostra la **gerarchia di memorie**, in cui il livello superiore (quello più vicino al processore) è il più piccolo e il più veloce:

- al livello più alto sono presenti i registri della CPU, con basso tempo di accesso e capacità di qualche byte;
- segue la memoria cache, che ha una capacità superiore a quella dei registri, ma un tempo di accesso più elevato;
- viene poi la RAM, che ha capacità maggiore della *cache*, ma anche tempo di accesso superiore;
- infine, al livello più basso, c'è la memoria di massa, che dispone di un'elevata capacità, ma con tempi di accesso che superano le RAM di parecchi ordini di grandezza.

→ Fig. 28 Gerarchia delle memorie: minore è il tempo di accesso, minore è la capacità.



Livelli adiacenti di memoria sono gestiti logicamente con la stessa modalità; quando la CPU richiede delle informazioni, accede alla memoria nel seguente modo:

1. accesso alla cache: se le informazioni vengono trovate sono portate nella CPU altrimenti le cerca nella memoria centrale;
2. accesso alla memoria centrale: se le informazioni vengono trovate sono copiate nella cache, altrimenti viene effettuato un accesso alla memoria di massa;
3. accesso alla memoria di massa: le informazioni devono essere trovate e copiate nella memoria centrale.

Lo scopo dei livelli superiori è quello di memorizzare quante più informazioni possibili in modo da evitare accessi ai livelli inferiori, che sono più costosi in termini di tempo e, proprio per questo, fanno diminuire le prestazioni. La scelta dell'insieme di informazioni da portare dal livello inferiore al livello superiore a esso adiacente è fatta in base ai principi di località.

### Pit Stop

- 20 Come è organizzata la gerarchia delle memorie di un sistema?
- 21 Che relazione c'è tra due livelli di memoria adiacenti?

## Comprendi con l'analogia

La gestione delle informazioni nella gerarchia delle memorie è come la gestione del processo domestico di rifornimento dell'acqua da bere

In molte famiglie si beve acqua minerale in bottiglia. Normalmente in cucina si tiene un cestino che può contenere poche bottiglie. Quando c'è bisogno di una bottiglia di acqua, si controlla se è disponibile in cucina: se c'è si preleva quella, altrimenti si va in cantina e si riempie un cestino intero. Infatti, dovendo fare le scale, non si preleva una sola bottiglia, ma tutte quelle che si possono trasportare, per non dover andare in cantina troppo spesso. Si cerca cioè di minimizzare l'accesso alla cantina, perché costa fatica e tempo. Ovviamente non è possibile portare tutte le casse di acqua in cucina, perché occuperebbero troppo spazio. Consideriamo adesso questa situazione: c'è bisogno di una bottiglia d'acqua, ma in cucina non c'è; e anche in cantina le bottiglie sono terminate. Non rimane allora che andare al supermercato a fare rifornimento. A questo punto non si acquista una sola cassa, ma nuovamente tutte quelle che si possono trasportare. Il tempo per andare al supermercato è enormemente maggiore di quello impiegato per andare in cantina, e bisogna perciò minimizzare anche questo.



## 20 I principi di località

Un'informazione viene cercata nei vari livelli di memoria, a partire da quello gerarchicamente più alto. Se il dato richiesto dal processore è presente in uno dei blocchi nel livello superiore, si dice che la richiesta ha successo (**hit**), altrimenti la richiesta fallisce (**miss**) e occorre quindi accedere a un livello inferiore della gerarchia.

Quando si accede a un livello inferiore, tuttavia, non viene prelevata solo l'informazione necessaria in quel momento, ma tutto un insieme di informazioni. Questo insieme ha un nome e una dimensione diversi, a seconda del livello di memoria interessato: dalla RAM alla cache viene trasferito un insieme di informazioni che viene chiamato **blocco**, dal disco alla RAM viene trasferito un insieme di informazioni che viene chiamato **pagina**.

I principi di località forniscono al sistema i criteri per la scelta delle informazioni da trasferire. I principi di località sono due:

- il **principio di località temporale**: se un dato viene richiesto in un certo istante, è probabile che lo stesso dato venga nuovamente richiesto entro breve (per esempio, quando il programma deve eseguire dei cicli). In questo caso si scelgono i dati usati più recentemente;

### Pit Stop

- 22 Che cosa si intende per «blocco»? E per «pagina»?

- il **principio di località spaziale**: se un dato è richiesto in un certo istante, è probabile che dati memorizzati in celle di memoria adiacenti vengano richiesti entro breve (per esempio, quando deve essere eseguito un programma sequenziale, o un accesso a vettori o matrici tramite un indice). In questo caso si scelgono i dati vicini a quello richiesto.

## 21 L'algoritmo LRU (Least Recently Used)

Quando occorre trasferire informazioni in un livello di memoria superiore (e quindi meno capace) lo spazio disponibile potrebbe non essere più sufficiente. In questo caso bisogna sovrascrivere una parte di memoria e, di conseguenza, perdere le informazioni che quella parte contiene. Esistono varie politiche per decidere quale area di memoria sovrascrivere. Uno degli algoritmi usati, a titolo di esempio, è l'algoritmo LRU (*Least Recently Used*, «il meno recentemente usato»): è lo stesso utilizzato dal sistema operativo per la gestione della memoria virtuale per simulare la disponibilità di una grande quantità di memoria RAM. Poiché la RAM ha capacità minore rispetto alla memoria di massa, si verifica il caso in cui è necessario trasferire una pagina di memoria, ma la RAM è piena. In questo caso il sistema operativo deve decidere quale pagina di memoria «perdere» per caricare quella necessaria.

Una delle politiche di gestione è appunto la LRU, in cui il sistema decide di perdere (sovrascrivere) la pagina meno recentemente usata.

### Pit Stop

23 Che cos'è l'algoritmo LRU?

### Comprendi con l'analogia

**La gestione della memoria con la politica LRU è simile alla gestione di un armadio di giacche pieno da cui, per far posto, si toglie la giacca che non si mette da più tempo**

L'algoritmo LRU agisce in modo simile al comportamento di una persona che dispone di un armadio per riporvi le giacche. Quando deve riporre nell'armadio pieno una giacca nuova, con quale criterio decide quale rimuovere per far posto? Un criterio potrebbe essere quello di togliere la giacca che non mette da più tempo (LRU), nell'ipotesi che non la metterà nel prossimo futuro. La giacca tolta non viene gettata o regalata: potrebbe essere riposta in un armadio in cantina; in tal caso, se dovesse riaverne bisogno, male che vada il proprietario della giacca impiegherà semplicemente più tempo per recuperarla.

## 22 Il controllo degli errori di memoria

Le memorie sono soggette all'azione di fenomeni fisici, come i raggi cosmici e le radiazioni ambientali, che possono alterarne il contenuto.

Fortunatamente, ci sono dei metodi matematici capaci di individuare gli errori che possono verificarsi. Queste tecniche sfruttano il concetto di **ridondanza**, ovvero l'aggiunta ai **bit di informazione** di **bit di controllo** utili a rilevare gli errori:

- se i bit di controllo sono in grado solamente di scoprire gli errori, sono chiamati **codici rilevatori di errori**;
- se i bit di controllo permettono anche la correzione, sono chiamati **codici rilevatori e correttori di errori**.

## I codici rilevatori di errori

Il meccanismo più comune utilizzato per **rilevare gli errori** è il **CRC (Cyclic Redundancy Check, «controllo di ridondanza ciclico»)** che consiste nell'associare a un blocco di dati un codice di controllo che valida i dati.

Il CRC opera un calcolo matematico (una divisione tra polinomi) su un insieme di dati e restituisce un numero (il resto della divisione) che caratterizza univocamente quel particolare blocco, come se fosse la sua impronta digitale. Se anche un solo bit varia, il CRC risulta diverso e l'errore viene rilevato, anche se non può essere corretto. Questa tecnica è applicata spesso alla memoria di massa e nella trasmissione dei dati.

## I codici rilevatori e correttori di errori

La **correzione degli errori** è affidata a codici correttori come, per esempio, quello di **Hamming**. Questa tecnica matematica, aggiungendo ai dati un certo numero di bit, consente di rilevare e correggere errori su un singolo bit. Il numero di bit utilizzati per la correzione aumenta all'aumentare del numero dei bit contenuti nel blocco dei dati. Per esempio, per correggere un blocco di 4 bit occorre aggiungerne altri 3 con un aumento del 75%.

### Esempio 6

#### Correzione a singolo bit

Con la **correzione dell'errore a singolo bit** è possibile correggere al massimo un bit per blocco di dati.

La **Figura 29a** mostra, sotto forma di insiemi, i 4 bit del blocco (o messaggio):

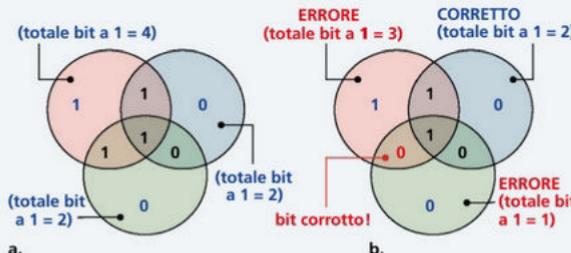
**Blocco dati** (4 bit): **1110** (in nero nel disegno)

Il codice di Hamming aggiunge **tre bit di parità** (in blu nel disegno) che sono ottenuti applicando la parità pari (operazione logica XOR) a tutte le possibili combinazioni di 3 bit estratti tra i 4 del messaggio (codice Hamming (7,4) che codifica i 4 bit del messaggio aggiungendo 3 bit di parità).

La figura sottostante fornisce una spiegazione intuitiva di come viene corretto un bit errato (visualizzato in rosso). Il numero dei bit a 1, compreso quello di parità, contenuti in ciascuno dei tre cerchi deve essere pari:

**Parità (3 bit): 100**

**Totale (dati + parità)** **1110 100**



Controllando la parità dei tre cerchi (**Figura 29b**) è possibile localizzare l'errore e correggerlo: il bit **0** che appartiene all'intersezione delle due regioni (di sinistra e di quella sottostante) è errato e va corretto con il valore **1** originale. Se i bit corretti sono più d'uno il processo non funziona.

## notabene

Le RAM che sono dotate di **ECC (Error Correction Code, «codice di correzione dell'errore»)** possiedono un sistema integrato, basato sul codice di Hamming, in grado di correggere immediatamente l'errore. Sono più costose e un po' più lente, ma sono più affidabili e sono principalmente utilizzate sui server e nei sistemi di comunicazione.

## notabene

Ricordiamo che si dice di «parità pari» un insieme di bit nel quale i bit uguali a 1 sono in numero pari.

◀ Fig. 29 a. Blocco originale: dati **1110**, con parità pari **100**; b. Blocco corrotto: dati **1100**, con parità pari **100** errata.

# SINTESI

## MEMORIA

È la componente del sistema in grado di memorizzare informazioni in binario. Si divide in memoria primaria (o centrale) e memoria secondaria. Si misura in byte.

### GERARCHIA DI MEMORIA

È in funzione del tempo di accesso e della capacità.  
Per aumentare le prestazioni si mantengono più informazioni possibili nei livelli più alti della gerarchia.  
Gerarchia: REGISTRI CPU – CACHE – RAM – SSD – HD – NASTRI.

### PRIMARIA

È la memoria direttamente collegata alla CPU.  
Si divide in RAM e ROM. La RAM è volatile (conserva i dati finché il chip è alimentato) ed è una memoria di lavoro. La ROM è permanente, in sola lettura, ed è utilizzata per l'avvio del sistema.

### SECONDARIA

È permanente (conserva i dati anche se il sistema non è alimentato) e di grandi dimensioni. Esempi di memoria secondaria sono i dischi HD o SSD.

**INDIRIZZO DI MEMORIA**

È la posizione di una cella di memoria e serve per identificarla.

**INDIRIZZO ASSOLUTO**

È la posizione di una cella di memoria a partire dalla prima, cioè quella di indirizzo 0.

**INDIRIZZO RELATIVO**

È la posizione di una cella di memoria rispetto a un'altra.  
Si parla in questo caso di base e spiazzamento.

**DECODIFICATORE**

Per attivare la cella di memoria indirizzata tramite l'address bus occorre un decodificatore, un circuito con  $n$  ingressi e  $2^n$  uscite.

**BUS DI SISTEMA****INTERAZIONE CPU-RAM**

L'esecuzione di un programma coinvolge il bus di sistema.  
Vengono attivati specifici segnali del control bus in base all'istruzione da eseguire.

**ESECUZIONE DI UN PROGRAMMA**

- fetch del codice operativo che si trova nella locazione puntata dal Program counter e viene portato nel registro IR tramite il bus dati;
- decode del codice operativo;
- caricamento di eventuali operandi;
- modifica del Program Counter aumentandolo in base alla lunghezza dell'intera istruzione.

**CONSIDERAZIONI SULLE DIMENSIONI DEL BUS**

Le dimensioni dei bus sono strettamente correlate alle dimensioni dei dati e degli indirizzi e incidono sulle prestazioni del sistema.

# Verifica delle CONOSCENZE

Unità 2



## Vero o Falso?

- 1 La RAM serve per la partenza del sistema.  V  F
- 2 L'architettura di Von Neumann prevede CPU, RAM, interfacce con le periferiche e bus di sistema.  V  F
- 3 La RAM è una memoria ad accesso diretto.  V  F
- 4 L'indirizzo di memoria identifica il contenuto di una locazione.  V  F
- 5 L'indirizzo assoluto è espresso con due numeri.  V  F
- 6 Lo spiazzamento è la distanza dalla base.  V  F
- 7 Con un bus indirizzi a 8 linee si possono indirizzare 255 byte.  V  F
- 8 Un puntatore è un indirizzo.  V  F
- 9 Il Program Counter punta alla cima dello stack.  V  F
- 10 Lo stack è una memoria.  V  F
- 11 La cache memory è una memoria permanente.  V  F
- 12 La memoria DRAM necessita di refresh.  V  F
- 13 Il segnale di refresh è una linea del bus di dati.  V  F
- 14 La EPROM è una memoria volatile.  V  F
- 15 La cache è utilizzata per migliorare le prestazioni del sistema.  V  F
- 16 La cache memory è una memoria permanente.  V  F
- 17 La gerarchia delle memorie è l'organizzazione delle cartelle in cui si salvano i file.  V  F
- 18 I registri della CPU sono più veloci e più capaci della cache memory.  V  F

- 19 DRAM e SRAM sono memorie permanenti.  V  F

- 20 Località spaziale significa che si deve trovare spazio nella memoria.  V  F

- 21 La RAM non è soggetta a errori di memorizzazione.  V  F

- 22 L'operazione POP è una lettura da memoria.  V  F

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

- 23 Comito principale della memoria centrale è:

- a fare evolvere i processi
- b contenere i nomi delle variabili di un programma
- c contenere i programmi in esecuzione (completi di dati e istruzioni)
- d eseguire istruzioni

- 24 Volendo definire una gerarchia di memoria in funzione della capacità (dalla minore alla maggiore), quale sequenza è corretta?

- a memoria centrale, registri, disco
- b la capacità delle memorie è sempre uguale
- c dipende dalle dimensioni della memoria centrale
- d registri, memoria centrale, disco

- 25 I principi di località

- a localizzano una cella di memoria
- b servono per decidere quali informazioni portare al livello di memoria superiore
- c sono applicati solo tra memoria volatile e memoria permanente
- d sono utili, ma rallentano il sistema

## Domande per la prova orale

- 26 Quali sono gli ordini di grandezza che definiscono la capacità dei diversi tipi memoria?
- 27 Che cosa è l'indirizzo di una locazione di memoria?
- 28 Che differenza c'è tra stack e stack pointer?
- 29 Quali operazioni possono essere fatte sullo stack?

- 30 Che cos'è un bit?

- 31 Che cos'è un byte?

- 32 Come si misura la capacità di memoria?

- 33 A che cosa corrisponde 1 kbyte?

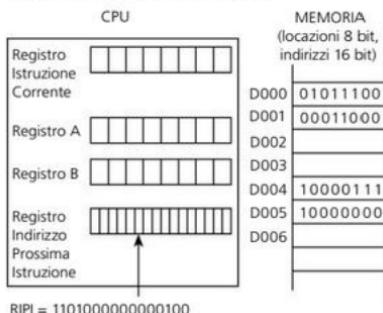
- 34 Qual è la funzione svolta dalla cache memory?

- 35 Che cos'è e come viene utilizzato lo stack?

# Verifica delle ABILITÀ

Unità 2

- 36 Considera la seguente situazione:



Esegui le seguenti istruzioni e rispondi alle domande.

- a. Prendi il dato contenuto nella locazione di indirizzo D000 e mettilo in RA.  
RA = \_\_\_\_\_
- b. Prendi il dato contenuto nella locazione di indirizzo D001 e mettilo in RB.  
RB = \_\_\_\_\_
- c. Prendi l'istruzione contenuta nella locazione di memoria il cui indirizzo è in RIPI e mettila in RIC.  
RIC = \_\_\_\_\_
- d. Supponendo che l'istruzione appena letta sia A – B, esegui l'istruzione. Se il risultato è minore di 0, metti il risultato nella locazione di memoria di indirizzo D002;

se è maggiore o uguale a 0, mettilo nella locazione di memoria di indirizzo D003 (i numeri in memoria sono rappresentati in complemento a 2).

- e. Somma 1 al contenuto di RIPI e metti il risultato in RIPI. Prendi l'istruzione contenuta nella locazione il cui indirizzo è in RIPI, e mettila in RIC.  
RIC = \_\_\_\_\_

37 Determina la dimensione minima di un bus indirizzi utile a indirizzare 8 kbyte.

38 Determina l'indirizzo minimo e l'indirizzo massimo delle locazioni di una memoria con capacità 256 kbit e locazioni di 1 byte.

39 Una memoria contiene complessivamente 232 bit. Quante celle da 1 byte si possono avere, e quante da 4 byte? (esprimi i calcoli mantenendo le potenze di 2)

40 Se l'ultima cella di una memoria ha indirizzo FFFFh e ogni cella è di 8 bit, quanti byte contiene la memoria? Quanti bit? Quante word? (parole di 2 byte)

## Risolvi i seguenti problemi.

- 41 Completa una tabella che confronta la DRAM con la SRAM, utilizzando almeno cinque parametri di confronto.
- 42 Effettua una ricerca sul web per stabilire quali sono le principali caratteristiche delle memorie RAM che influenzano le prestazioni.



Esercizi in più



## The memory

### Abstract

The memory is the computer component dedicated to contain data and instructions. There are several memory types classified according to their characteristics; an important memory feature is the ability to retain the stored data when the system is powered-off.

Depending on their capacity and access speed, the memories are hierarchically classified: register, cache, RAM, mass storage; at top we find faster memories (expensive) and at bottom there are

slower but higher capacity memories (cheaper). In order to improve its performance, the system must try to maintain data into the higher ("nearest to the CPU") and faster memory levels, reducing the access to slower memories.

The CPU executes the basic program instructions reading them from the main memory (RAM); the stack is the RAM part that holds parameters and local variables.

### Questions

#### Answer all questions

- 1 What is locality of reference (also known as *the principle of locality*)?
- 2 What are the consequences of memory physical limits?
- 3 What does volatile memory mean?
- 4 What is a pointer?
- 5 How memory capacity is measured?

#### Choose the correct answer



- 6 Select the right memory hierarchy:
- [a] central memory, registers, cache, non volatile memory
  - [b] registers, cache, central memory, non volatile memory

- [c] cache, registers, central memory, non volatile memory
  - [d] non volatile memory, registers, cache, central memory
- 7 The stack:
- [a] is managed by a FIFO policy
  - [b] is a memory area containing instructions
  - [c] is a memory area used for data in special situations
  - [d] always starts at address 0
- 8 A memory location address:
- [a] indicates the content type
  - [b] indicates the position of the memory location
  - [c] is loaded in the current instruction register
  - [d] always has the same size as the referred location one

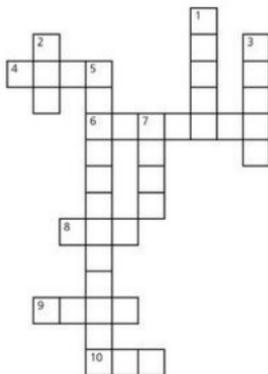
### Crosswords

#### Across

- 4 Happens when information sought after in the memory is not found
- 6 Identifies a position in memory
- 8 A read operation from the stack
- 9 Measurement unit for memory
- 10 Read-only memory

#### Down

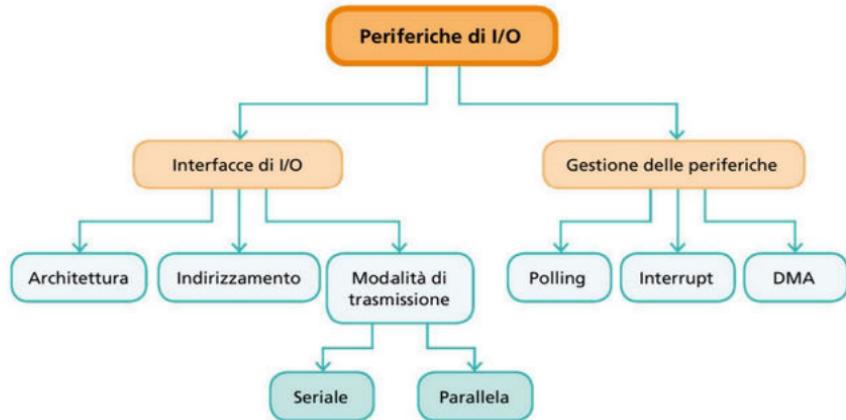
- 1 Is a buffer between the CPU and the main memory
- 2 Codifies the physical state of a signal
- 3 Memory that contains data
- 5 Contains the address to the top of the stack
- 7 Needs refreshing





# Unità 3

## Le periferiche di I/O



### Visione d'insieme

- Le interfacce di I/O per comunicare con l'esterno del sistema.
- Architettura di un sistema di I/O.
- Tecniche per la gestione delle periferiche di I/O.

## 1 Introduzione

Un computer deve poter interagire con il mondo esterno.

La comunicazione con le grandezze fisiche dell'ambiente e con gli esseri umani è realizzata attraverso le **interfacce di ingresso/uscita**.

Si tratta di circuiti hardware che hanno la funzione di collegare il data bus del computer con le periferiche.

Ciò è necessario per colmare la distanza tra il mondo analogico, rappresentato da grandezze fisiche continue, e il mondo digitale basato su grandezze discrete.

I computer hanno leggi proprie, diverse da quelle del mondo ordinario. Al loro interno i dati marcano su un bus come soldatini ben allineati. Il ritmo di lavoro è imposto da un orologio che batte il tempo a una frequenza altissima e i valori di tensione e corrente possono essere molto diversi da quelli utilizzati nella vita quotidiana.

### Comprendi con l'analogia

#### I tempi della realtà esterna rispetto a quelli di un computer sono come quelli di un bradipo rispetto ai nostri

I bradipi sono animali famosi per la loro lentezza. Si muovono a una velocità di qualche centimetro al secondo e possono trascorrere l'intera vita su un albero.

Per quanto riguarda i tempi di risposta, un computer avrebbe nei nostri confronti la stessa sensazione che proviamo noi verso i bradipi: nel tempo in cui un umano preme un tasto della tastiera, il computer vive una vita!



## 2 Le periferiche

### Pit Stop

- Che cos'è un'interfaccia di input/output?
- Quali sono le funzioni delle periferiche?

↓ Tab. 1 Classificazione delle periferiche più comuni

Una Ferrari è un'auto meravigliosa. Il rombo del suo motore è musica per le orecchie di chi lo apprezza. Eppure senza il volante e il cambio sarebbe solo un bel monumento all'ingegno umano.

La stessa cosa vale per uno smartphone, un PC o un supercalcolatore: senza le periferiche non sapremmo cosa farcene.

Le **periferiche** sono i dispositivi che permettono lo scambio di informazioni tra il computer e il mondo esterno.

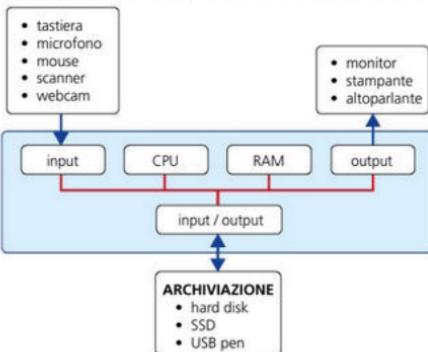
Hanno tre funzioni principali (**Tabella 1**):

- permettono di colloquiare con l'utente (monitor, tastiera ecc.);
- permettono di interagire con il mondo fisico (sensori, attuatori ecc.);
- permettono di immagazzinare le informazioni su memorie permanenti (dischi magnetici e a stato solido, nastri ecc.).

Funzione	Esempi di periferiche
Colloquio con l'utente	<ul style="list-style-type: none"> <li>• monitor touchscreen, tastiera, mouse, tavoletta grafica</li> <li>• stampanti inkjet e laser</li> <li>• videocamere e fotocamere</li> <li>• casse audio, cuffie</li> <li>• gamepad, joystick, sensori di movimento</li> </ul>
Interazione con il mondo fisico	<ul style="list-style-type: none"> <li>• apparati per la trasmissione dei dati in rete (modem)</li> <li>• sistemi elettromeccanici (relè, motori ecc.)</li> <li>• convertitori analogico-digitale e convertitori digitale-analogico</li> </ul>
Memorizzazione permanente di informazioni	<ul style="list-style-type: none"> <li>• dischi a stato solido, dischi magnetici (hard disk)</li> <li>• dischi ottici (CD, DVD)</li> <li>• nastri magnetici</li> </ul>

Dal punto di vista del trasferimento dei dati, le periferiche si possono classificare come dispositivi di **input**, in cui le informazioni transitano dall'esterno verso il data bus del computer, e dispositivi di **output**, in cui le informazioni transitano dal data bus del computer verso l'esterno (**Figura 1**).

Le prime comprendono, per esempio, tastiera, mouse, touchscreen, microfono e sensori, le seconde monitor, stampanti, modem, sistemi audio, attuatori. Alcune periferiche, come i dischi o le schede di rete per la trasmissione dei dati, sono sistemi **bidirezionali** (input/output), perché possono sia inviare sia ricevere informazioni.



### Pit Stop

- 3 Che cosa significa che una periferica è bidirezionale?

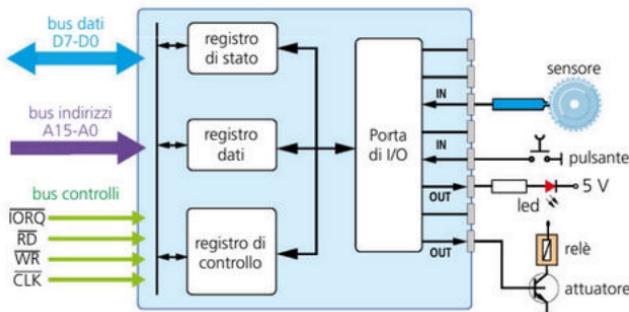
◀ Fig. 1 Le periferiche di ingresso e uscita più comuni. Le periferiche per l'immagazzinamento delle informazioni sono bidirezionali (input/output), perché i loro dati possono essere sia scritti sia letti.

Le periferiche possono essere:

- **esterne** all'elaboratore, cioè collegate al sistema tramite cavi che si inseriscono nelle porte dell'interfaccia. Una porta connette elettricamente e meccanicamente una periferica con l'interfaccia;
- **interne** all'elaboratore, cioè collegate al sistema tramite *bus* interni al case del computer.

## 3 Le interfacce

Il punto di incontro tra le periferiche e il *bus* del sistema è costituito dalle **interfacce**, sistemi hardware e software, che utilizzano metodi standard per l'indirizzamento, il controllo, la sincronizzazione e lo scambio di dati con le periferiche. La **Figura 2** mostra la struttura generale di una interfaccia, che comprende:



◀ Fig. 2 Architettura di una generica interfaccia di Ingresso/Uscita. La parte di sinistra comprende il buffer di dati, il registro di stato e il registro di controllo necessario per la programmazione del dispositivo.

- una o più **porte di uscita e/o di ingresso** per lo scambio dei segnali con la periferica;
- un **registro di controllo** che gestisce il funzionamento delle porte dell'interfaccia, in cui ogni pin può essere programmato in ingresso o in uscita;
- un **registro di stato**, che indica la situazione attuale della periferica (per esempio, attesa dati, dato presente, ...);
- un **registro dati (buffer)**, con il valore da trasferire in uscita o ricevuto in ingresso.

Ogni registro ha un indirizzo con cui viene identificato. Il microprocessore accede ai registri con le **istruzioni IN e OUT**.

## 4 L'indirizzamento

L'indirizzamento dei registri associati a una porta di ingresso o di uscita è simile a quello visto precedentemente per le memorie, con la differenza che l'indirizzo che viene posto sull'address bus non è quello di una cella di memoria, bensì quello di una porta. Il segnale IORQ è utilizzato per abilitare l'interfaccia.

L'esempio seguente (**Figura 3**) mostra i segnali attivati per effettuare la lettura/scrittura di dati tra CPU e interfaccia di I/O:

- la lettura da parte della CPU di un dato dall'interfaccia si ottiene con l'attivazione contemporanea della linea di IORQ e RD;
- la scrittura da parte della CPU di un dato sulla periferica si ottiene con l'attivazione contemporanea dei segnali IORQ e WR.

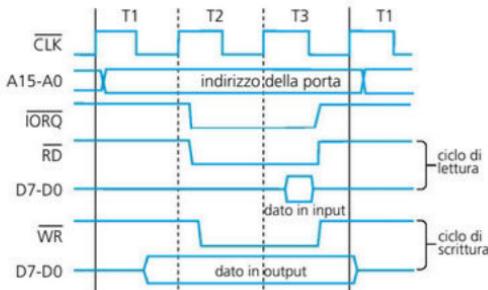
### ■ notabene

Un segnale «attivo basso» è valido quando il suo stato è uno 0 logico. Graficamente il suo segnale compare soprassegnoato.

→ **Fig. 3** Una generica lettura da una interfaccia di I/O prevede l'attivazione della linea RD (attivo basso). La scrittura di un dato in memoria si serve del segnale WR (attivo basso). In entrambi i casi il segnale IORQ (attivo basso) diventa attivo quando il valore dell'indirizzo della cella di memoria, presente sul bus indirizzi, è stabile.

### ■ Pit Stop

4 Che cosa si intende per «memory mapped»?



Un altro modo per accedere a un'interfaccia di I/O è quello di considerarla come una locazione di memoria (*memory-mapped*). In questo caso vengono riservati indirizzi diversi da quelli delle celle di memoria e il processore esegue operazioni di ingresso/uscita come se fossero normali azioni di lettura/scrittura in memoria senza servirsi delle istruzioni di IN e OUT.

### Esempio 1

L'indirizzamento delle porte di I/O potrebbe essere fatto seguendo esattamente lo schema mostrato nell'Unità 2 inerente la decodifica degli indirizzi con la differenza che la linea del *control bus* che abilita il decodificatore (ingresso di *Enable*) non è MREQ, ma IORQ, che è attivato dalle istruzioni di IN e OUT.

Consideriamo lo schema di **Figura 4** che comprende due ipotetiche unità di interfaccia.

I **chip** sono raggiunti dalle uscite **Ux** del decodificatore che abilita i due **Chip Select** a fronte delle linee di indirizzo **A15 e A14** che possono indirizzare fino a 4 dispositivi.

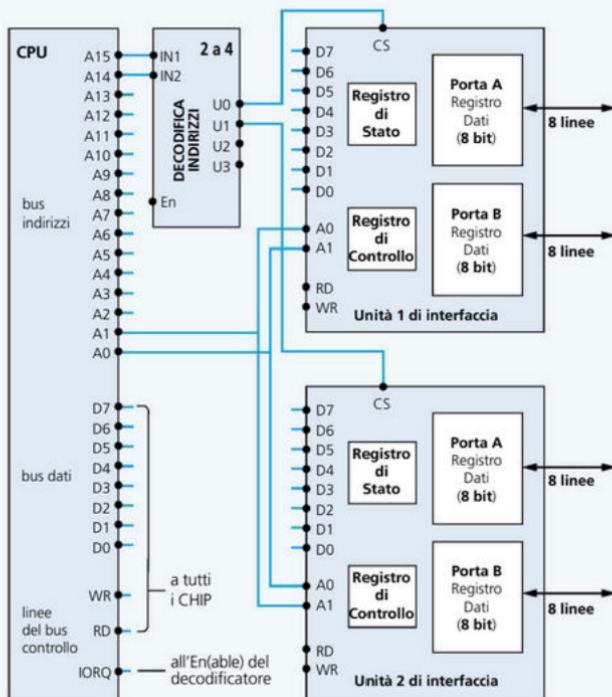


Fig. 4 Schema del collegamento alle due unità di interfaccia, ciascuna contenente due porte dati, una porta di controllo e una di stato.

Ciascun chip di interfaccia comprende:

- i **registri dati** (Porta A e Porta B a 8 bit);
- un **registro di controllo** in sola scrittura, che serve per definire quali sono le linee di ingresso e di uscita;
- un **registro di stato** in sola lettura.

Per indirizzare i quattro registri di ogni interfaccia occorrono le due linee **A0** e **A1** collegate agli ingressi del dispositivo.



L'accesso ai registri interni di ciascuna unità di interfaccia è presentato in **Tabella 2**.

RD	WR	A1	A0	Registro	Lettura/scrittura
1	0	0	0	Porta A	Read
0	1	0	0	Porta A	Write
1	0	0	1	Porta B	Read
0	1	0	1	Porta B	Write
1	0	1	0	Stato	Read
0	1	1	0	—	—
1	0	1	1	—	—
0	1	1	1	Controllo	Write

† **Tab. 2** Tabella di indirizzamento dei registri.

## 5 La trasmissione dei dati

Particolari interfacce sono quelle utilizzate per la trasmissione dei dati. Poiché la trasmissione può essere seriale o parallela (per esempio, le reti di computer e le comunicazioni a lunga distanza utilizzano la comunicazione seriale, che è più economica), anche le interfacce sono di due tipi: interfacce seriali e interfacce parallele (**Figure 5a e 5b**).

### Interfaccia seriale

Trasmette i dati come una serie di impulsi di tensione lungo un unico collegamento. A ogni impulso di *clock* l'interfaccia seriale invia, uno dopo l'altro, i bit del dato: la sequenza temporale determina il valore binario.

#### Interfaccia seriale asincrona

La sincronizzazione del trasmittitore e del ricevitore è garantita dai bit di start e stop che precedono e seguono il dato trasmesso.  
Nella **Figura 5a** è mostrata una trasmissione seriale asincrona, in cui gli 8 bit del dato (10100101) sono preceduti dal bit di start e seguiti dal bit di stop. Questi due bit servono al ricevitore per sincronizzarsi.

#### Interfaccia seriale sincrona

I bit vengono trasmessi come un flusso continuo, senza bit di start né bit di stop. Il ricevitore estrae il clock di sincronizzazione sfruttando le transizioni 1 e 0 dei bit dei dati in arrivo (che, in alternativa, può essere fornito da una linea separata).

### Interfaccia parallela

Utilizza più fili di collegamento su ciascuno dei quali vengono trasmessi, in contemporanea, ad ogni colpo di *clock*, i bit. In questo modo i dati possono essere inviati molto più velocemente rispetto alla trasmissione seriale, ma con costi di cablaggio decisamente superiori.

### notabene

Molti protocolli di collegamento che sfruttano la trasmissione sincrona prevedono la trasmissione continua di bit di sincronizzazione (flag), anche quando il collegamento è inattivo.

La **Tabella 3** sintetizza le interfacce per la trasmissione dati più utilizzate nei computer.

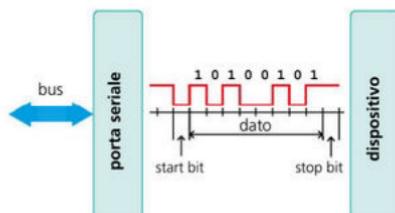
Interfacce parallele e seriali	
<b>Interfacce seriali</b>	
<ul style="list-style-type: none"> <li>• <b>seriali tradizionali</b> (RS-232, RS-422, RS-485) a 9 o a 25 pin per il collegamento a <i>modem</i> o ad altri <i>computer</i> o dispositivi (velocità fino a circa 100 kbps). Ha avuto enorme importanza negli anni passati, ora è utilizzata molto più raramente;</li> <li>• <b>USB (Universal Serial Bus)</b>, veloci e molto popolari, per il collegamento fino a 127 periferiche, che possono essere collegate anche con il computer acceso;</li> <li>• <b>IrDA (Infrared Data Association)</b>, collegamento a infrarossi senza contatto fisico;</li> <li>• <b>FireWire</b> (IEEE 1394), per trasferimento dati tra due dispositivi;</li> <li>• <b>Ethernet</b>, per reti locali;</li> <li>• <b>HDMI (High-Definition Multimedia Interface)</b>, per i collegamenti audio e video;</li> <li>• <b>SATA (Serial ATA)</b>, per il collegamento di dischi o lettori ottici.</li> </ul>	
<b>Interfacce parallele</b>	
<ul style="list-style-type: none"> <li>• <b>PPI (Parallel Peripheral Interface)</b>: originariamente nei PC sono state utilizzate per le stampanti, oppure come un'interfaccia per la gestione di singoli ingressi/uscite per sensori/attuatori;</li> <li>• <b>SCSI (Small Computer System Interface)</b>: interfaccia standard per il trasferimento di dati su bus, ormai obsoleta, perché più lenta e complessa rispetto a tecnologie più recenti, come SATA. La sua evoluzione è <i>Serial Attached SCSI (SAS)</i> che però è un'interfaccia seriale.</li> </ul>	

### notabene

I pin, o piedini, sono le terminazioni elettriche su cui passano i segnali che permettono a un circuito integrato di collegarsi con l'esterno. Per convenzione, i pin di un circuito integrato sono numerati in senso antiorario, a partire dal pin in alto a sinistra.

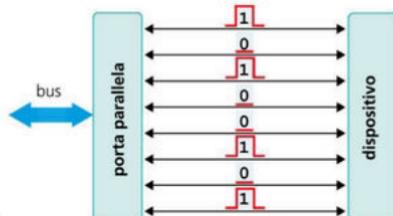
◀ Tab. 3 Interfacce parallele e seriali.

#### Collegamento seriale asincrono (1 bit alla volta)



a.

#### Collegamento parallelo (8 bit)



b.

◀ Fig. 5

a. trasmissione seriale asincrona;

b. trasmissione parallela.

## notabene

Nei comuni sistemi operativi presenti sui PC la gestione delle periferiche necessita di un dispositivo hardware (controller) e di software specifici (driver software) capaci di controllare il funzionamento dell'hardware delle periferiche. Lo standard Plug&Play permette ai dispositivi di autoconfigurarsi.

## Pit Stop

- 5 Qual è il maggior limite del polling?

## 6 Tecniche per la gestione delle periferiche

La CPU può utilizzare tre diverse tecniche per comunicare con una interfaccia di *Input/Output* e trasferire i dati:

- **Polling**: la CPU interroga ciclicamente la periferica;
- **Interrupt**: la periferica interrompe la CPU per segnalare che è avvenuto un evento;
- **DMA (Direct Memory Access)**: la periferica accede direttamente alla memoria centrale senza coinvolgere la CPU.

### ■ Polling: la periferica è controllata dalla CPU

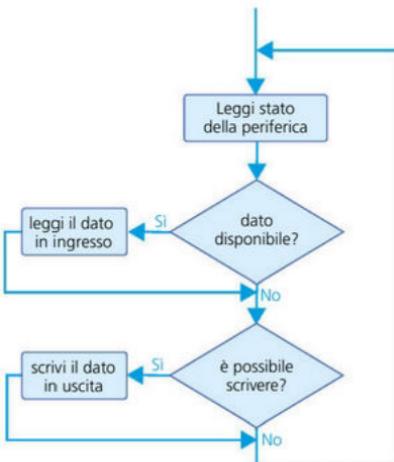
La tecnica del **polling** consiste in una interrogazione ciclica e sequenziale che la CPU effettua nei confronti delle periferiche (**Figura 6**).

L'unità centrale legge il registro di stato dell'interfaccia per controllare se il dato è disponibile e, quindi, leggerlo con una istruzione di *input*.

Per esempio è possibile stabilire quando un tasto viene premuto sulla tastiera ed effettuare la lettura del codice.

Il limite maggiore di questa tecnica consiste nel fatto che la CPU è costretta a spendere gran parte del suo tempo a interrogare le periferiche, anche se queste non hanno nulla da segnalare.

È come se guardassimo continuamente il display del telefono per controllare se è arrivato il messaggio che stiamo aspettando.



→ **Fig. 6** Schema del funzionamento della gestione di periferica tramite polling. La CPU interroga ciclicamente le periferiche.

### ■ Interrupt: la periferica interrompe la CPU

La tecnica dell'interrupt consiste nell'interruzione, da parte di una periferica, del programma della CPU. La CPU sospende l'esecuzione del programma in corso per eseguire una «routine», cioè un blocco di codice richiamabile dal programma principale, al termine della quale riprende il programma interrotto nel punto

in cui l'aveva lasciato (**Figura 7**). In questo modo la CPU può rispondere immediatamente a una richiesta della periferica. Per questo motivo è utilizzata nella gestione dei processi in tempo reale e nella ricezione e trasmissione dei dati. La tecnica dell'interrupt è una tecnica più efficiente del polling, anche se più complessa perché richiede l'aggiunta di un hardware esterno.



### Pit Stop

- 6 A che cosa serve un interrupt?
- 7 Che cos'è una routine?

Fig. 7 Schema del funzionamento della gestione di periferica tramite interrupt. La periferica «chiama» la CPU, che passa all'esecuzione delle routine di servizio.

## Comprendi con l'analogia

L'interrupt è come l'interruzione dell'attività di un cuoco che esegua una ricetta di cucina

Consideriamo un uomo che sta preparando una torta seguendo una ricetta in un libro di cucina.

Immaginiamo che, in un momento non prevedibile, la figlia irrompa in cucina piangendo e dicendo di essere stata punta da un'ape mentre giocava in giardino. L'uomo segna il punto della ricetta in cui è arrivato (lo stato del processo in esecuzione viene salvato), prende il manuale di pronto soccorso e comincia a seguirne le indicazioni. Medicata la figlia, l'uomo riprenderà a cucinare la torta, dal punto in cui era stato interrotto.



Gli interrupt possono essere hardware o software.

**Interrupt hardware.** La periferica invia alla CPU un segnale di interruzione utilizzando una linea del control bus.

È un interrupt asincrono, perché l'istante in cui perviene alla CPU la richiesta di interruzione non è prevedibile. Gli interrupt hardware possono essere di due tipi:

- **Interrupt mascherabile (Maskable Interrupt).**  
La richiesta di interruzione perviene alla CPU su una linea del control bus, chiamata spesso **INT** o **INTR**. La richiesta di interruzione può essere ignorata (mascherata) dalla CPU. L'interrupt mascherabile può essere utilizzato per la gestione di situazioni non particolarmente critiche come, per esempio, la ricezione di un dato da tastiera o l'invio di un dato sulla porta seriale.
- **Interrupt non mascherabile (Non-Maskable Interrupt).**  
La richiesta di interruzione perviene alla CPU su una linea del control bus (spesso chiamata **NMI**). La richiesta di interruzione **non può** essere ignorata

### Pit Stop

- 8 Che differenza c'è tra un interrupt mascherabile e uno non mascherabile?

dalla CPU che deve obbligatoriamente servire la richiesta. Queste interruzioni sono generate da eventi critici, come errori di memoria irrecuperabili, cali di alimentazione e altre situazioni non prevedibili.

### ■ notabene

Nei sistemi X86 l'istruzione per la generazione di un interrupt software è INT.

**Interrupt software.** L'interruzione è generata dal programma durante la sua esecuzione senza l'intervento esterno di una periferica. Gli *interrupt software* possono avvenire in due modi:

- tramite **istruzioni speciali di tipo sincrono (INT)**, che spesso sono usate per comunicare con il sistema operativo della macchina;
- causati da anomalie (**Exception**) o errori di **esecuzione del programma**, come la divisione per zero o lo sfondamento della memoria di stack.

La Figura 8 riassume la classificazione degli *interrupt*.

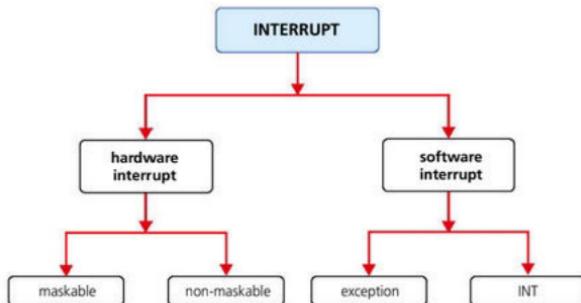


Fig 8 Classificazione delle interruzioni.

### ■ Pit Stop

- 9 Qual è il maggior vantaggio di gestire una periferica in modalità DMA?

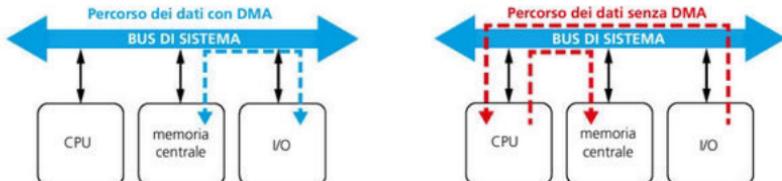
Fig. 9 La comunicazione tra le periferiche di interfaccia e la memoria centrale può avvenire secondo due modalità, con DMA e senza DMA.

### ■ DMA: la periferica controlla direttamente la memoria

Il **Direct Memory Access** è utilizzato per il trasferimento diretto dei dati tra I/O e memoria, senza dover passare dalla CPU: la periferica assume il controllo diretto delle operazioni di lettura e scrittura in memoria.

Il dispositivo DMA si sostituisce alla CPU, prende in carico il *bus* e fa transitare i dati direttamente dalla periferica (per esempio la memoria di massa) alla memoria centrale, e viceversa.

L'accesso diretto alla memoria permette di operare il trasferimento dei dati alla massima velocità. In questo caso, sullo stesso *bus* si posizionano due elementi principali: la CPU e il dispositivo che controlla il DMA. Solo uno dei due elementi deve essere attivo: mentre il controllore del DMA prende il controllo, la CPU si «stacca» dal *bus* (passa nello stato di «HOLD») per ritornare attiva alla fine della procedura di trasferimento diretto (Figura 9).



# Approfondimento - Tecnologia

## Un esempio di gestione degli interrupt

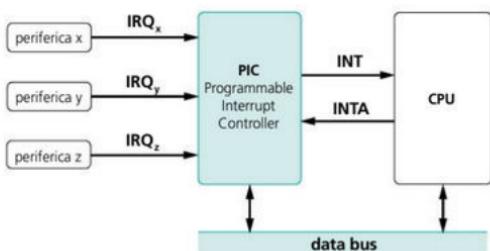
Nella **Tavola 1** sono precise le operazioni svolte dalla CPU a fronte di un segnale di interrupt.

In base all'architettura del microprocessore le azioni intraprese e le linee del *control bus* coinvolte nel processo possono essere diverse, ma il comportamento della CPU è sostanzialmente il medesimo. Nell'esempio in tabella facciamo riferimento al meccanismo di **Interrupt ReQuest (IRQ)**, tipico dei sistemi X86 presenti nei PC, in cui un hardware esterno alla CPU, il «controllore programmabile delle interruzioni» (**PIC, Programmable Interrupt Controller**) convoglia i segnali di interrupt che arrivano dalle diverse periferiche (tastiera, porte seriali ecc.) in un'unica uscita collegata alla linea di INT del microprocessore.

Interrupt mascherabile	Interrupt non mascherabile
<ul style="list-style-type: none"> <li>La CPU riceve, sulla <b>linea di INT</b>, la richiesta di interruzione da parte di una periferica o dell'hardware che fa da tramite («controllore programmabile delle interruzioni»).</li> <li>La CPU controlla lo stato del <i>flag</i> di interrupt (IF), presente nel registro di stato.</li> <li>Se IF è attivo, cioè se la CPU è abilitata ad accettare l'interruzione, la CPU comunica al dispositivo la propria disponibilità a servire l'interrupt attivando la linea del control bus INTA (<b>Interrupt Acknowledge</b>) che, in uscita dalla CPU, è collegato al controllore delle interruzioni.</li> <li>Il controllore invia alla CPU sul bus dati il codice identificativo della periferica che ha fatto richiesta.</li> <li>La CPU salva lo stato del processo in esecuzione e inizia a eseguire la routine di servizio associata alla periferica che ne ha fatto richiesta.</li> <li>Al termine della routine la CPU ripristina lo stato del processo interrotto e prosegue la sua esecuzione dal punto in cui era stato interrotto.</li> </ul>	<ul style="list-style-type: none"> <li>La richiesta di interruzione perviene alla CPU sulla <b>linea dedicata NMI</b> del control bus.</li> <li>La CPU salva lo stato del processo in esecuzione e inizia a eseguire la routine di servizio predefinita, associata all'interrupt non mascherabile e posta a un indirizzo di memoria prestabilito senza la necessità di identificare la periferica.</li> <li>Al termine della routine la CPU ripristina lo stato del processo interrotto e prosegue la sua esecuzione dal punto in cui era stato interrotto.</li> </ul>

↑ Tab. 1

È possibile che più periferiche cerchino di interrompere contemporaneamente la CPU o che la routine di risposta all'interrupt venga interrotta a sua volta. Per risolvere questo problema si possono adottare diverse soluzioni hardware e software, ma la scelta più comune è quella di delegare questo compito al PIC che viene programmato per ordinare le interruzioni, assegnando a esse le priorità (**Figura 1**).



↑ Fig. 1 Il PIC assegna la priorità alle periferiche che hanno inviato un interrupt alla CPU.

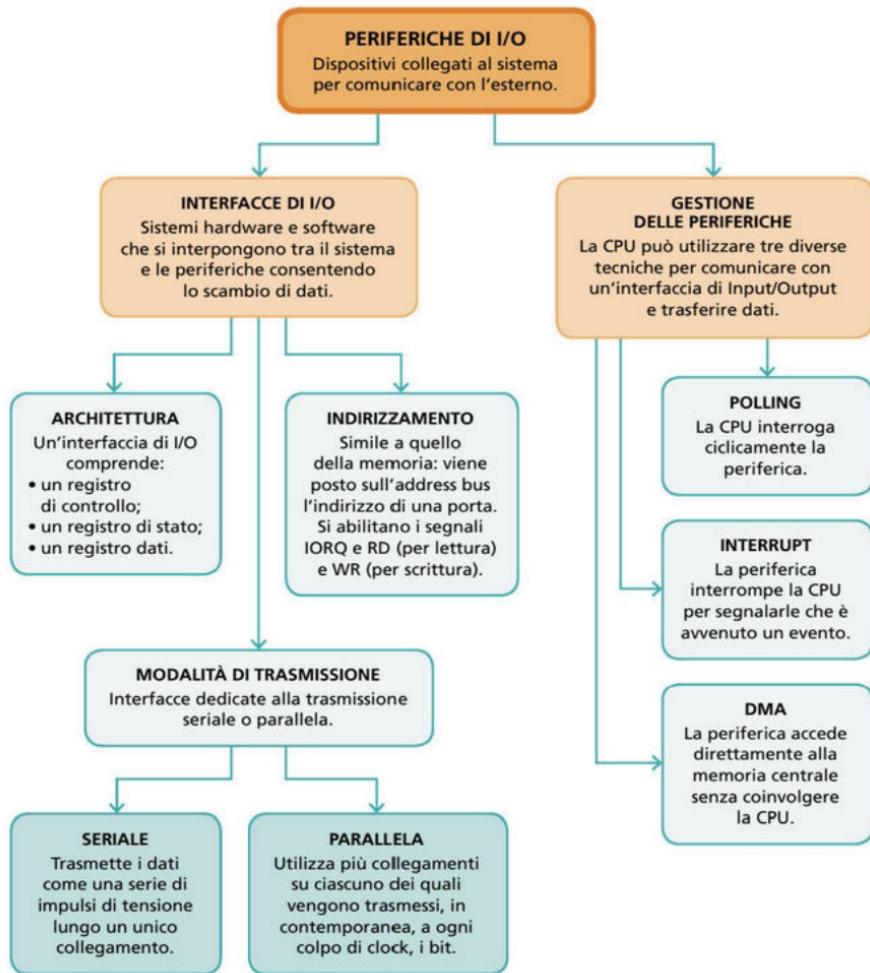
### Comprendi con l'analogia

La linea dedicata NMI è come una linea telefonica con un numero per le emergenze, la linea INT è come una linea telefonica con un numero per le informazioni

La situazione delle due possibili linee di interrupt è simile a quella di una clinica veterinaria che ha due linee telefoniche relative a due numeri diversi: la linea associata agli ambulatori, il cui numero è quello da chiamare per ottenere informazioni e prenotare le visite periodiche, e quella associata al pronto soccorso veterinario, da chiamare in caso di emergenza. Quando squilla il telefono dell'ambulatorio, il centralinista può decidere se rispondere o no. Se accetta di rispondere, alza il ricevitore, identifica chi ha chiamato ed evade la richiesta. Se invece squilla il telefono del pronto soccorso, sa già che si tratta di un'emergenza: risponde e allerta immediatamente lo staff.

Un normale ciclo di lavoro può essere interrotto da chiamate ordinarie (telefono dell'ambulatorio) oppure da chiamate inderogabili e non mascherabili (telefono del pronto soccorso).





# Verifica delle CONOSCENZE

Unità 3



## Vero o Falso?

- 1 Le periferiche di I/O fanno parte del modello di Von Neumann.  
  F
- 2 Le interfacce di I/O fanno parte del modello di Von Neumann.  
  F
- 3 Un'interfaccia di I/O è un dispositivo programmabile.  
  F
- 4 Il registro di stato di un'interfaccia contiene l'istruzione inviata dalla CPU.  
  F
- 5 Il polling consiste in un'interrogazione ciclica e sequenziale.  
  F
- 6 L'interrupt consiste in un'interrogazione ciclica e sequenziale.  
  F
- 7 L'interrupt hardware è un segnale del bus controllo.  
  F
- 8 Quando la CPU riceve un segnale di interrupt, interrompe l'esecuzione del processo in corso a favore di una routine di servizio.  
  F
- 9 DMA è una periferica di Output.  
  F
- 10 La tastiera è una periferica di Input/Output.  
  F
- 11 La porta USB è una periferica universale.  
  F
- 12 Una lettura da periferica abilita i segnali IORQ e MREQ.  
  F
- 13 Tramite la porta parallela può essere effettuata la trasmissione di un byte alla volta.  
  F
- 14 Memory-mapped considera una interfaccia di I/O come una locazione di memoria.  
  F
- 15 Per indirizzare una interfaccia di I/O occorre un decodificatore.  
  F

- 16 Il microprocessore scrive su una periferica tramite istruzione OUT.  
  F

- 17 Il microprocessore legge da periferica tramite istruzione OUT.  
  F

- 18 Le porte USB permettono una trasmissione molto veloce e possono connettere fino a 127 periferiche, oltre all'unità di controllo.  
  F

- 19 Un interrupt hardware è sincrono.  
  F

- 20 La tecnica dell'interrupt è più efficiente del polling.  
  F



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

- 21 Un driver:
  - a) è l'hardware associato a una periferica
  - b) è il software associato a una periferica
  - c) è un'interfaccia di I/O
  - d) nessuna delle precedenti risposte è vera
- 22 Un interrupt:
  - a) è un segnale inviato dalla CPU a una periferica
  - b) è un'istruzione eseguita dalla periferica
  - c) è un segnale inviato da una periferica alla CPU
  - d) è un'interrogazione ciclica
- 23 Per una lettura di dati dall'esterno sono abilitati i segnali del control bus:
  - a) MREQ, RD
  - b) IOREQ, RD
  - c) IOREQ, WR
  - d) INTA, RD

- 24 Come è possibile mettere in relazione il mondo digitale (veloce e temporizzato) del calcolatore con il mondo fisico esterno alla macchina (lento e analogico)?

- 25 Che differenza c'è tra trasmissione seriale e parallela? E tra trasmissione seriale sincrona e asincrona?

- 26 Quali sono le modalità di interazione tra CPU e periferiche?

- 27 Che cosa si intende per Input?

- 28 Che cosa si intende per Output?

- 29 Che differenza c'è tra un'interfaccia di I/O e una periferica di I/O?

- 30 Descrivì l'architettura interna di una generica interfaccia di I/O.

- 31 Quali sono i motivi per cui è necessario collegare le periferiche tramite interfacce?

- 32 Che differenza c'è tra trasmissione seriale e trasmissione parallela?

- 33 Esponi le tecniche studiate per la gestione delle periferiche.

# Verifica delle ABILITÀ

Unità 3

- 34 Fai almeno tre esempi di periferica di Input.
- 35 Fai almeno tre esempi di periferica di Output.
- 36 Elenca vantaggi e svantaggi della gestione delle periferiche tramite interrupt.
- 37 Classifica le seguenti periferiche (tra Input, Output, Input/Output):

*tastiera mouse scanner microfono trackball touchpad webcam monitor altoparlanti stampante plotter cuffie modem*

Input
Output
Input/Output

- 38 Riassumi, utilizzando la tabella, le caratteristiche delle tecniche di gestione delle periferiche.

	Polling	Interrupt	DMA
Descrizione			
Vantaggi			
Svantaggi			

- 39 Considera un sistema con 4 porte di I/O. Completa la tabella specificando i possibili valori IORQ, RD, WR e bit per indirizzamento (linee A0 e A1) per effettuare operazioni di lettura e scrittura da ciascuna porta.

RD	WR	A1	A0	IORQ	Registro	Lettura/scrittura

- 40 Fai una ricerca sul web sull'evoluzione delle porte USB. Sintetizza i dati raccolti in una tabella, specificando le fonti da cui hai tratto le informazioni. Correda il documento di immagini.
- 41 Fai una ricerca sul web sulle interfacce di rete (NIC). Sintetizza i dati raccolti in una tabella, specificando le fonti da cui hai tratto le informazioni. Correda il documento di immagini.



Esercizi in più

## The Input/Output devices

### Abstract

A computing system communicates with the external world by I/O devices; the I/O interfaces connect the I/O devices to the central system. Each interface has its internal architecture that, if the device is programmable, will include data, control and status registers.

Some basic I/O management techniques are:

- Polling - the CPU periodically checks the status of an I/O device to know when it's ready to read or write; with this technique the CPU 'wastes' its



time, continuously checking the devices status;

- Interrupt - the device sends an interrupt to the CPU when it needs to do something; only at this point the CPU interacts with the device;
- Direct Memory Access (DMA) - when a device needs to transfer data to/from a memory, the CPU just starts the device activity; the device directly manages the data transfer and, when finished, sends an interrupt to the CPU, to go on.

### Questions

#### Answer all questions

- 1 What are the I/O peripherals for?
- 2 What does *interrupt controlled I/O* mean?
- 3 Write some examples of input device.
- 4 What are ports?

#### Choose the correct answer



- 5 A serial port:
  - [a] permits self-configuring devices
  - [b] is a device that transfers data one bit at a time
  - [c] is a device that transfers data in parallel
  - [d] is an output device

#### 6 The polling:

- [a] is a technique for device management
- [b] is an I/O interface
- [c] is used for I/O devices programming
- [d] interrupts the CPU

#### 7 Device interfaces:

- [a] are useful because the CPU speed is different from the devices speed
- [b] are user friendly
- [c] increase system performances
- [d] are used for CPU programming

#### 8 A programmable device:

- [a] has status, data and control registers
- [b] has a RAM
- [c] has a CPU
- [d] is an algorithm

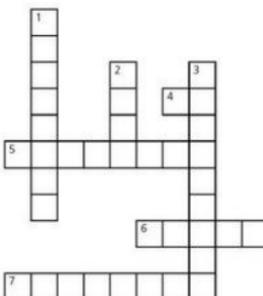
### Crosswords

#### Across

- 4 An instruction used to read data from a port.
- 5 An Interrupt that can be ignored.
- 6 A pointing input device.
- 7 A networking system used in local networks (LAN).

#### Down

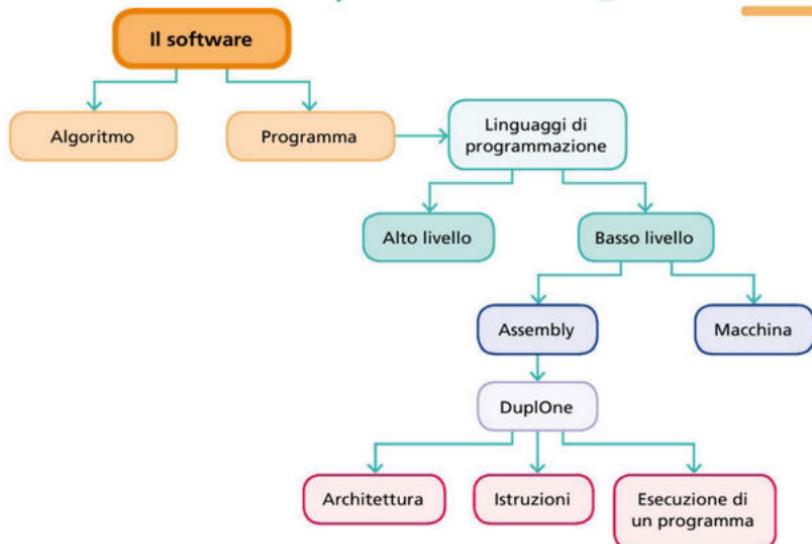
- 1 Standard input device.
- 2 A contactless infrared link.
- 3 A technique used to manage I/O devices.





# Unità 4

## Il software (con DuplOne)



### Visione d'insieme

- Il software di un sistema di elaborazione.
- Classificazione dei linguaggi di programmazione.
- Il linguaggio Assembly.

## 1 Introduzione

Hardware significa letteralmente «ferramenta». Senza un programma che lo istruisce sui compiti da eseguire, un computer è poco più di un ferro rigido e freddo. Il software è costituito dai programmi che fanno funzionare l'hardware. L'importanza del software è aumentata col passare del tempo fino a offuscare la visione del processore, che rimane comunque il vero centro nevralgico di un computer.

Quando siamo alle prese con un computer in funzione è come se avessimo a che fare con un'auto a guida autonoma: ne percepiamo le funzionalità e l'intelligenza di guida e solo in un secondo momento pensiamo al motore che la muove.

La collaborazione tra hardware e software è quella che decreta il successo di ogni nuova applicazione. Per questo è bene conoscere l'unico **linguaggio** che l'elaboratore conosce: il **codice macchina**, in cui ogni istruzione è codificata in una sequenza di bit che indica le operazioni da eseguire e gli operandi su cui agire. Ci sono molti **linguaggi di programmazione** nati per fare da ponte tra il linguaggio umano e quello numerico delle macchine, ma tutti, per essere compresi dal calcolatore, devono essere tradotti nel codice macchina.

### Pit Stop

- Che cos'è il linguaggio o codice macchina?
- Qual è la differenza tra algoritmo e programma?

## 2 Che cos'è un programma

Occorre distinguere il significato di *programma* da quello di *algoritmo*:

- un **algoritmo** è una sequenza finita di istruzioni volte a risolvere un problema e scritte in un linguaggio «naturale», chiamato anche *pseudocodice*;
- un **programma** è un algoritmo codificato utilizzando un determinato linguaggio di programmazione, cioè un linguaggio che può essere «capito» dalla CPU.

Prima di scrivere un programma è necessario definire l'algoritmo, cioè l'idea per la soluzione del problema dato. Poiché uno stesso problema può essere risolto in modi diversi, possono esistere diversi algoritmi per la soluzione. Una volta definito l'algoritmo, occorre scegliere il linguaggio più opportuno per implementarlo. L'algoritmo può anche essere scritto su un pezzo di carta, il programma invece deve essere scritto e salvato su un file, chiamato **file sorgente**.

### Ogni algoritmo può essere codificato con diversi linguaggi

Partiamo dall'esempio seguente per osservare che uno stesso algoritmo può essere implementato con diversi linguaggi di programmazione.

### La parola a... Muhammad ibn Mūsā al-Khwārizmī

Il termine *algoritmo* deriva dal nome del matematico musulmano Muhammad ibn Musa, vissuto a Bagdad, nell'attuale Iraq, tra il 780 e l'850 d.C., e chiamato al-Khwarizmi, cioè «originario della Corasmia». Il trattato di algebra (*al-jabr*), la sua opera più importante, lo rese tanto famoso da essere ricordato attraverso paesi e secoli come il «trattato degli algoritmi», cioè dei procedimenti di calcolo insegnati da al-Khwarizmi.

Si racconta che alla richiesta del califfo di occuparsi di scienze diverse dalla matematica egli rispose: «Adesso penso solo a una cosa, ovvero al modo di facilitare lo studio della matematica a tutta la gente. È inutile studiare una scienza che non è utile nella vita pratica». Infatti, a proposito del suo lavoro, scriveva che il suo intento era quello di «comporre una breve opera sul calcolo [...] limitandosi a quegli aspetti più facili e utili della matematica di cui si serve costantemente nei casi di eredità, donazioni, distruzioni, sentenze e commerci e in tutti gli altri affari umani, o quando si vogliono effettuare misurazioni di terreni, scavi di canali, calcoli geometrici e altre cose del genere».

## Esempio 1

### Problema

Stampare a video «ciao mondo» per 5 volte.

### Algoritmo scritto con pseudocodice

```
i = 0
mentre (i < 5) esegui
    scrivi "ciao mondo"
    i = i + 1
fine mentre
Istruzione_successiva
```

### Codifica in linguaggio di programmazione C

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
    int i=0;
    while (i<5) {
        printf ("ciao mondo");
        i++;
    }
    return 0;
}
```

### Codifica in linguaggio di programmazione PHP

```
<?php
$i=0;
while ($i < 5) {
    print ("ciao mondo");
    $i +=1;
}
?>
```

### Codifica in linguaggio di programmazione Python

```
i=0
while i<5:
    print ("ciao mondo")
    i+=1
```

### Codifica in assembly 8086

```
        mov i, 0
Whileloop: cmp i, NUM
            jge WhileEnd
            lea dx, var_ciao;
            stampa "ciao mondo"
            mov ah, 9
            int 21h
            inc i
            jmp Whileloop
WhileEnd:
            lea dx, pkey
            mov ah, 9
            int 21h
            mov ah, 1
            int 21h
            mov ax, 4c00h
            int 21h
ends
```



**notabene**

La sequenza di bit dell'ultimo frammento di programma, scritto in linguaggio macchina, è pura invenzione. Quello che si vuol fare notare è che il linguaggio macchina è codificato come sequenze binarie.

**Codifica in Java**

```
int i;
i=0;
while (i < 5) {
    System.out.println("Ciao mondo");
    i++;
}
```

**Codifica in linguaggio macchina**

```
000010100100100100010101000101001001000100010001010001000
10101010100101001000101000101001010010001010101001000
101010101011111111110101010100100010001111100101010100
10101000101010101010.....(e molti altri ancora)
```

Possiamo fare delle **osservazioni** che ci consentono di dividere i linguaggi usati nell'esempio in due gruppi.

Programmi scritti in C, PHP, Python, Java	Programmi scritti in Assembly (e linguaggio macchina)
<ul style="list-style-type: none"> <li>• Più simili allo pseudocodice;</li> <li>• più facili da capire;</li> <li>• più brevi;</li> <li>• utilizzano strutture di controllo (nell'esempio il <code>while</code>).</li> </ul>	<ul style="list-style-type: none"> <li>• Più difficili da capire;</li> <li>• dipendenti dall'hardware; infatti nelle istruzioni sono presenti i nomi dei registri (<code>ah, dx, ...</code>);</li> <li>• non ci sono strutture di controllo;</li> <li>• è specificato il nome dell'architettura (nell'esempio <code>8086</code>).</li> </ul>

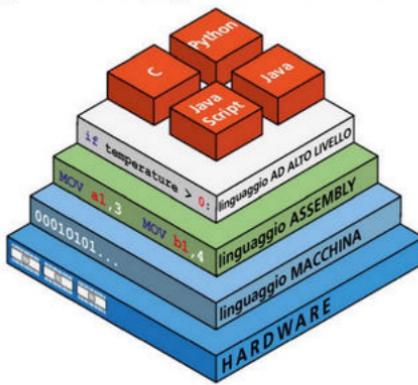
I linguaggi del primo gruppo sono **linguaggi ad alto livello**, quelli del secondo gruppo sono **linguaggi a basso livello**.

### 3 Linguaggi a basso e ad alto livello

**Pit Stop**

- 3 Qual è la principale differenza tra un linguaggio ad alto livello e uno a basso livello?

→ Fig. 1 Linguaggi a basso e ad alto livello: livelli più bassi sono più vicini all'hardware; cioè più dipendenti dall'hardware; livelli più alti sono più astratti, più indipendenti dall'hardware.



## I linguaggi a basso livello

I linguaggi a basso livello sono diversi per ogni CPU. Sono facilmente interpretati dall'hardware, ma sono ostici per gli esseri umani:

- Il linguaggio di livello più basso è il **linguaggio macchina**, o **codice macchina**, in cui ogni istruzione è codificata come una sequenza di simboli 0/1.
- Appena sopra il linguaggio macchina si pone l'**Assembly**, un linguaggio intermedio costituito da codici mnemonici, cioè facili da ricordare, che rendono la lettura del codice più chiara.
- A ogni istruzione Assembly corrisponde una istruzione in linguaggio macchina. Ogni architettura ha il proprio set di istruzioni. Un programma scritto in Assembly necessita di un programma assemblatore (**Assembler**) che traduce il codice Assembly in linguaggio macchina (**Figura 2**).
- Una locazione di memoria è identificata tramite il suo indirizzo.

### Codice Assembly

	mov A, [LOCAZIONE1]
	mov B, [LOCAZIONE2]
	mov [LOCAZIONE1],A
	add A,B
	sub A,B
	jnz, etichetta
	halt
etichetta	inc A
64h	
65h	

L'assemblatore traduce il programma Assembly e produce il codice macchina

### Codice macchina in RAM

```
000100110001001000011
1100001001100011001100
010010001000010011000
110010010000010011111
111101000001
```

Fig. 2 La traduzione delle istruzioni Assembly da parte di un programma assemblatore produce il codice macchina in formato binario.

## I linguaggi ad alto livello

I linguaggi ad alto livello (per esempio C, Pascal, Java, Python...) sono più vicini al linguaggio umano, ma sono lontani dall'architettura fisica dell'elaboratore.

- Un'istruzione scritta in linguaggio ad alto livello corrisponde a più istruzioni scritte in un linguaggio a basso livello.
- Aumenta il livello di astrazione e migliora la leggibilità dei programmi con ricadute positive sulla sua manutenzione e condivisione.
- Il compito di tradurre un linguaggio di alto livello in linguaggio macchina è svolto dal compilatore o dall'interprete.
- Nei linguaggi ad alto livello vengono utilizzate le variabili. Una variabile è l'astrazione di una cella di memoria e ha un nome e un valore. Il nome di una variabile (alto livello) corrisponde all'indirizzo della cella (basso livello). Usando una variabile il programmatore non è tenuto a conoscere l'indirizzo della locazione di memoria. L'associazione tra variabile e memoria è effettuata dal compilatore.

### Pit Stop

- Come viene codificata un'istruzione in linguaggio macchina?
- Che cos'è l'Assembler?
- Che cos'è il compilatore?
- Che cos'è una variabile?

La **Tavola 1** confronta i linguaggi ad alto e a basso livello, evidenziando le loro caratteristiche.

→ **Tab. 1**

Linguaggi ad alto livello	Linguaggi a basso livello (macchina/Assembly)
<ul style="list-style-type: none"> <li>• «Astragno» dall'architettura dell'elaboratore;</li> <li>• offrono una scelta tra molti linguaggi;</li> <li>• la loro esecuzione è più lenta;</li> <li>• hanno istruzioni potenti, semplici e generali;</li> <li>• sono facili da imparare e di pronto uso, con molte librerie di programmi disponibili;</li> <li>• sono utilizzati per la realizzazione delle applicazioni software più comuni.</li> </ul>	<ul style="list-style-type: none"> <li>• Dipendono dall'hardware e dall'architettura;</li> <li>• la loro esecuzione è più veloce;</li> <li>• c'è una corrispondenza 1:1 tra istruzioni in Assembly e istruzioni in linguaggio macchina;</li> <li>• richiedono tempi lunghi e uno sforzo maggiore per l'apprendimento e per la realizzazione dei programmi nel linguaggio specifico del processore.</li> </ul>

## La parola a... Ada Byron Lovelace, la madre di tutti i programmatori

Ada Byron Lovelace (1815-1852) può essere considerata la «madre di tutti i programmatori». Fu lei che cento anni prima che il calcolatore fosse inventato ebbe l'intuizione che «rendere un meccanismo capace di combinare simboli generali, in sequenze illimitate per varietà ed estensione, vuol dire avere trovato un punto di unione fra le operazioni della materia e i processi astratti della mente». Ada prefigura una macchina che può rappresentare, oltre ai numeri, qualsiasi elemento simbolico della realtà, come testi e musica. Non solo: Ada immagina la capacità di apprendere delle macchine, anticipando le soluzioni di Intelligenza Artificiale e mostrandone i limiti: «la macchina analitica non ha alcuna pretesa di concepire alcunché», ma solo di eseguire i compiti che noi le assegniamo.



## Comprendi con l'analogia

### I codici di un computer sono come quelli della natura

Dal diario di Giulia Mercuri che realizza applicazioni web.

Tasera, a tavola, mia figlia di terza superiore mi chiede: «Oggi a scuola parlavano di DNA, il codice della vita. C'entra qualcosa con i linguaggi dei computer?» La domanda mi sorprende e non so rispondere.

«Mangia che si raffredda!» taglio corto e cambio discorso, anche se la cosa mi incuriosisce.

Dopo cena provo a fare una ricerca in rete e... scopro delle somiglianze stupefacenti tra il DNA e il codice binario, a me più familiare.

Nella memoria del computer gli «0» e gli «1» sono memorizzati in forma di cariche elettriche, mentre in natura le informazioni sono immagazzinate come catene del DNA.

L'informazione, in un computer, è codificata con i simboli binari, mentre l'informazione biologica è trasmessa da una successione di 4 simboli (**Figura 3**).

Il computer possiede un disco a stato solido. Il DNA è come un supporto di memoria ridondante e affidabile. Per elaborare un programma, il computer deve trasferirlo dal disco alla RAM, mentre il DNA si serve dell'RNA messaggero.



GAGAA <del>V</del> AAAC	ΨAGΨΑVVVΨΨΨΨ	CΨGGΨCC <del>CC</del> CA	CAGACΨCAGA	GAGAACC <del>CC</del> GC	50
CACCA <del>V</del> GΨΨC	ΨΨGΨΨCCΨΨG	ΨΨGΨΨCΨΨC	ΨΨGΨΨGΨΨC	AGCCAGΨΨGΨ	100
ΨΨA <del>CC</del> CΨΨGAC	CACCA <del>G</del> AACA	CAGCΨWCCΨC	CAGCCΨACAC	CAACAGCΨΨΨ	150
ACCAGAGGCG	ΨΨGΨΨACΨΨCC	CGACA <del>AGG</del> GΨ	ΨΨCAGAΨΨCC	GCGΨΨCΨΨGCA	200
CΨΨΨΨ <del>CCCC</del> CAG	GACCCΨΨΨΨCC	ΨΨGΨΨΨΨΨΨΨΨ	CAGCΨA <del>CC</del> GΨ	ACCΨΨΨΨΨΨΨΨC	250
ACGCCAΨΨCCA	CGΨΨGΨΨCCGCG	ACCAΨΨGCA	CCAAGAGAΨΨΨ	CGACA <del>AC</del> CCC	300
ΨΨGΨΨΨΨΨΨΨΨ	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	GCCAGΨΨΨΨΨΨΨΨ	AGAAGΨΨΨΨΨΨΨΨA	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	350
ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	AGΨΨΨΨΨΨΨΨΨΨΨΨΨΨA	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	400
ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	AGΨΨΨΨΨΨΨΨΨΨΨΨΨΨA	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	450
ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	ACAAGAΨΨΨΨΨΨΨΨA	ΨΨΨΨΨΨΨΨΨΨΨΨΨΨ	500

↑ Fig. 3 Un esempio di una sequenza dei 4 simboli che trasmettono l'informazione biologica: i primi 500 caratteri del codice del vaccino mRNA BNT162b2 (Fonte: World Health Organization).

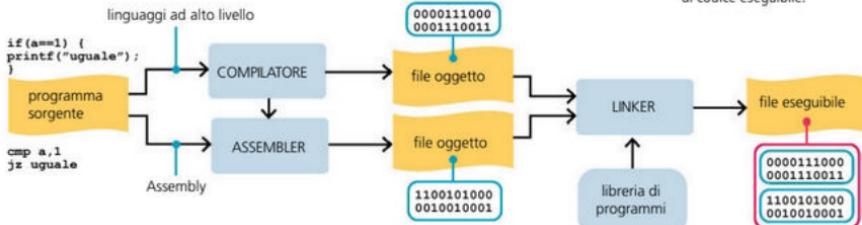
## 4 Dal codice sorgente al codice eseguibile

Un programma è scritto e salvato su un file, ma per essere eseguito deve essere tradotto in binario e caricato in memoria (Figura 4).

I software di base si occupano di tradurre un codice da codice sorgente a codice eseguibile:

- il **codice sorgente** è salvato su un file: *file sorgente*;
- il **compilatore** (o Assembler) effettua l'analisi sintattica del file sorgente e lo traduce in binario, generando il *file oggetto*;
- il **linker** «cuce» insieme diversi file oggetto in un unico file, generando il *file eseguibile* che è caricato in memoria per essere eseguito.

↓ Fig. 4 Dal codice sorgente al codice eseguibile.



Un **compilatore** effettua l'analisi sintattica di ogni istruzione, costruisce la **symbol table** (tabella dei simboli) che associa i nomi delle variabili agli indirizzi di memoria, e **genera un file oggetto** nel codice macchina del processore.

Un **interprete** effettua l'analisi sintattica dell'istruzione che, se corretta, viene subito eseguita in tempo reale. La traduzione di ogni istruzione comporta una maggiore lentezza nell'esecuzione rispetto a un programma che viene compilato.

### notabene

Un esempio di Symbol Table è presente in diverse attività di laboratorio.

Per alcuni linguaggi esiste sia un interprete sia un compilatore. Per esempio:

- C e C++ sono linguaggi compilati, perché usano un compilatore;
- Javascript è un linguaggio interpretato;
- Java utilizza un compilatore che traduce il programma sorgente in un linguaggio intermedio (bytecode), che poi viene interpretato dalla Java Virtual Machine.

## 5 Il linguaggio Assembly

Studiare il linguaggio Assembly è importante per capire veramente come è fatto e come funziona un sistema di elaborazione. Usare l'Assembly è l'unico modo per interagire direttamente con la macchina.

Una generica istruzione in linguaggio macchina è formata da due parti:

<codice operativo> e <operandi>

Il **codice operativo** (*operation code*) indica l'istruzione che la CPU deve eseguire, gli **operandi** specificano i dati su cui l'istruzione deve operare.

Nel linguaggio Assembly il codice operativo del linguaggio macchina viene sostituito da un codice mnemonico che ricorda il significato dell'istruzione. Per esempio, il codice operativo della somma può essere rappresentato con add, il codice della differenza con sub, un salto con jmp e un trasferimento dati con mov. Un'istruzione può avere uno o più operandi, oppure non averne.

In presenza di due operandi uno è la sorgente, l'altro la destinazione, cioè dove viene posto il risultato. Ogni Assembly ha la sua sintassi.

Consideriamo una istruzione Assembly con due operandi, in cui il primo è la destinazione e il secondo la sorgente:

```
mov A,4
```

### Pit Stop

- 8 Qual è la forma di una generica istruzione in linguaggio macchina?
- 9 Che cosa si intende per «codice mnemonico» in Assembly?

Il suo significato è: trasferisci il valore 4 (sorgente) nel registro A (destinazione). In genere un set di istruzioni Assembly è diviso in categorie in base al tipo di istruzione:

- istruzioni di trasferimento;
- istruzioni aritmetico-logiche;
- istruzioni di salto;
- istruzioni di I/O.

## 6 Il microprocessore DuplOne e il suo linguaggio Assembly

Esistono molti linguaggi Assembly, uno per ogni architettura, ma inizieremo lo studio del linguaggio a partire da un **microprocessore** appositamente inventato dagli Autori del corso per scopi didattici: il **DuplOne**. Il suo set di istruzioni è pensato esclusivamente per capire il funzionamento di base di un microprocessore, ma è verosimile, nel senso che utilizza istruzioni simili a quelle dei microprocessori che si trovano in commercio.

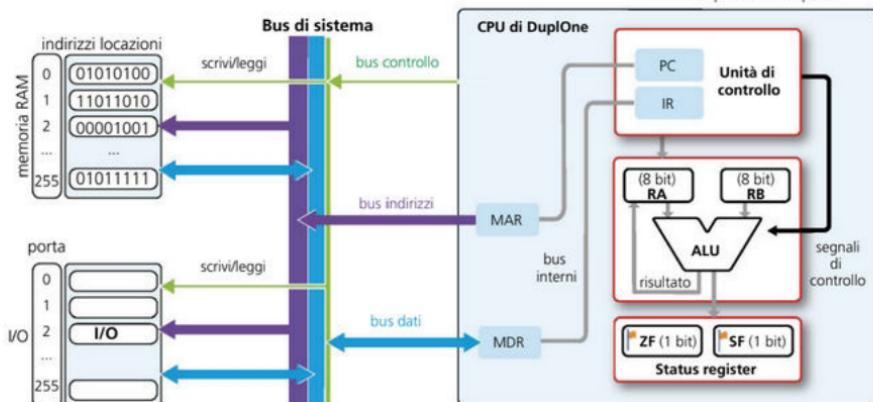
In seguito prenderemo in considerazione linguaggi Assembly reali come quello che sta alla base della famiglia x86, relativa ai microprocessori Intel, e il linguaggio dei microprocessori ARM che è attualmente il più diffuso, soprattutto nei dispositivi mobili e nelle single board come Raspberry Pi.

### Architettura di DuplOne

L'architettura del microprocessore DuplOne è ridotta all'essenziale e prevede ([Figura 5](#)):

- **bus:**
  - bus dati a 8 bit;
  - bus indirizzi a 8 bit;
  - bus controllo;
- **registri:**
  - due registri di uso generale a 8 bit: registro A (accumulatore) e registro B;
  - Status Register con Flag di Zero, Flag di Segno;
- **organizzazione della memoria:**
  - locazioni di memoria di 1 byte;
  - è indirizzabile il singolo byte;
  - memoria indirizzabile:  $2^8 = 256$  byte.

↓ Fig. 5 L'architettura del microprocessore DuplOne.



### Il linguaggio Assembly DuplOne

Il codice operativo di ogni istruzione nel linguaggio Assembly DuplOne è formato da 8 bit.

Le istruzioni hanno lunghezza di 1 o 2 byte:

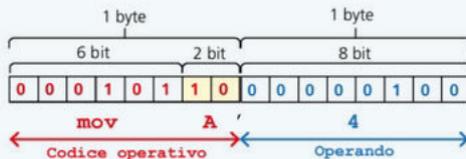
- le istruzioni lunghe 1 byte non hanno come operandi indirizzi di memoria o numeri;
- le istruzioni lunghe 2 byte hanno 1 byte per il codice operativo e un byte che specifica un indirizzo o un numero.

### Esempio 2

- L'istruzione `mov A, 4` corrisponde al codice macchina:

```
00010110  
00000100
```

- Il primo byte è il codice operativo, in cui i 6 bit più significativi (000101) corrispondono a `mov` e i restanti (10) indicano la destinazione (registro A)
- il secondo byte è il valore da trasferire ([Figura 6](#)).

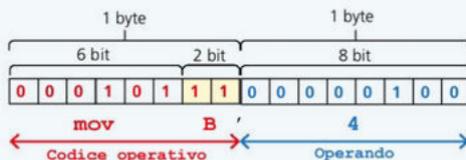


↑ Fig. 6 Codice operativo e operandi dell'istruzione `mov A, 4`. L'operando A è implicito nel codice operativo.

- L'istruzione `mov B, 4` corrisponde al codice macchina:

```
00010111  
00000100
```

- Il primo byte è il codice operativo in cui 000101 corrisponde a `mov` e 11 indica che la destinazione è il registro B;
- il secondo byte è il valore da trasferire ([Figura 7](#)).



↑ Fig. 7 Istruzione `mov B, 4` in Assembly DuplOne e in codice macchina.

### Esecuzione di un frammento di programma con il microprocessore DuplOne

L'esempio che segue ci aiuta a capire in che modo il microprocessore DuplOne preleva una istruzione dalla memoria centrale, la interpreta e la esegue seguendo le fasi di *fetch*, *decode* ed *execute*.

DuplOne «non ha idea» di che cosa eseguirà, se un calcolo o un confronto o un trasferimento di dati. L'unica cosa che «sa» con certezza è che **dovrà prendere dalla memoria il byte (istruzione) indirizzato dal Program Counter, lo interpreterà ed eseguirà l'istruzione associata.**

Consideriamo un frammento di programma che occupa 7 byte di memoria ed è costituito da 4 istruzioni.

Per l'interpretazione dei codici facciamo riferimento al set delle istruzioni mostrato nella **Tavella 2** del Paragrafo 7. Per ora limitiamoci a considerare solo quelle che ci servono per l'interpretazione di questo frammento di programma.

Indirizzo	Codice operativo/Operando	Istruzione	Commento
00000000 00000001	00010011 00010010	mov A, [12h]	Carica nel registro A il contenuto della locazione di indirizzo 00010010.
00000010 00000011	00011110 00010011	mov B, [13h]	Carica nel registro B il contenuto della locazione di indirizzo 00010011.
00000100 00000101	00011001 00010010	mov [12h], A	Carica nella locazione di indirizzo 00010010 il contenuto del registro A.
00000110	00100001	add A, B	Somma il contenuto del registro A al contenuto del registro B e pone il risultato della somma nel registro A.

### notabene

Ricordiamo che un numero seguito da h è scritto in base esadecimale.

Ad esempio:  
 $12h = 10010$  (base 2) = 18 (base 10)

Alla partenza (*Reset*), il PC (Program Counter) è caricato con il valore binario 00000000 che corrisponde all'indirizzo di memoria in cui è presente la prima istruzione del programma

$$\text{PC} = 0$$

## ISTRUZIONE 1

### • FETCH

Il codice operativo 00010011 contenuto nella locazione di memoria puntata da PC, viene prelevato e trasferito in IR.

### • DECODE

Di quale istruzione si tratta? Per rispondere occorre consultare la tabella del set di istruzioni assembly di DuplOne. Il codice operativo corrisponde a quello dell'istruzione:

mov A, [indirizzo]

che significa: trasferisci nel registro A il contenuto della locazione di memoria di indirizzo specificato.

Quindi 00010011 00010010 significa: «carica in A il valore posto all'indirizzo 12h».

Questa istruzione occupa 2 byte: uno per il codice operativo e uno per l'indirizzo di memoria che contiene l'operando.

Prima della fase di execute viene aggiornato il PC in modo che contenga l'indirizzo della prossima istruzione da eseguire:

$$\text{PC} = \text{PC} + \text{lunghezza dell'istruzione corrente}$$

Nel caso specifico l'istruzione corrente occupa 2 byte, e quindi:

$$\text{PC} = \text{PC} + 2 = 0 + 2 = 2$$

### ■ **notabene**

In tutte le operazioni di lettura/scrittura sono attivati segnali sulle corrispondenti linee del bus controllo.

- **EXECUTE**

Viene trasferito nel registro A il dato presente nella locazione 12h.

### **ISTRUZIONE 2**

- **FETCH**

Trasferisci nel registro IR il codice operativo 00011110...

- **DECODE**

... che corrisponde all'istruzione:

```
mov B,[indirizzo]
```

che viene così interpretata: «trasferisci nel registro B il contenuto della locazione di memoria di indirizzo specificato». In questo caso: «trasferisci in B il valore posto all'indirizzo 13h».

Questa istruzione occupa 2 byte: uno per il codice operativo e uno per l'indirizzo di memoria che contiene l'operando.

Prima della fase di execute viene aggiornato il PC in modo che contenga l'indirizzo della prossima istruzione da eseguire:

$$\text{PC} = \text{PC} + 2 = 2 + 2 = 4$$

- **EXECUTE**

Il dato presente nella locazione 13h è trasferito nel registro B.

### **ISTRUZIONE 3**

- **FETCH**

Trasferisci in IR il codice operativo 00011001...

- **DECODE**

... che corrisponde all'istruzione:

```
mov [indirizzo] , A
```

Che viene così interpretato: «trasferisci nella locazione di indirizzo specificato il contenuto del registro A».

In questo caso «trasferisci nella locazione di memoria di indirizzo 12h il contenuto del registro A».

Prima della fase di execute viene aggiornato il PC in modo che contenga l'indirizzo della prossima istruzione da eseguire:

$$\text{PC} = \text{PC} + 2 = 4 + 2 = 6$$

- **EXECUTE**

L'istruzione viene eseguita, viene trasferito il contenuto del registro A nella locazione di memoria di indirizzo 12h.

## ISTRUZIONE 4

- **FETCH**

Trasferisci nel registro IR il codice operativo 00100001...

- **DECODE**

che corrisponde all'istruzione:

add A,B

che viene così interpretato: «somma il contenuto del registro B al contenuto del registro A e poni il risultato in A».

Questa istruzione occupa 1 byte. Nel codice 00100001 è codificata sia l'operazione da fare, sia l'operando sorgente sia il destinatario.

Prima della fase di execute viene aggiornato il PC in modo che contenga l'indirizzo della prossima istruzione da eseguire:

$$\text{PC} = \text{PC} + 1 = 6 + 1 = 7$$

- **EXECUTE**

La alu somma al contenuto del registro A il contenuto del registro B e pone il risultato in A.

## 7 Il set di istruzioni del linguaggio Assembly DuplOne

Il set di istruzioni del linguaggio Assembly DuplOne è composto da istruzioni lunghe uno o due byte, raggruppate in categorie che rispecchiano la loro funzione (**Tabella 2**):

- istruzioni di trasferimento;
- istruzioni aritmetico-logiche;
- istruzioni di salto;
- istruzioni che coinvolgono lo stack;
- istruzioni di Ingresso/Uscita;
- istruzioni generali.

↓ Tab. 2

#	Istruzione Assembly - mnemonica	Codice Operativo (codice macchina binario)	hex	dec	Descrizione
<b>Istruzioni di trasferimento dalla memoria in un registro</b>					
1	<b>mov A,[indirizzo]</b>	0001 0011 nnnnnnnn (indirizzo)	13	19	Carica nel registro A il contenuto della locazione di memoria di indirizzo specificato
2	<b>mov B,[indirizzo]</b>	0001 1110 nnnnnnnn (indirizzo)	1E	30	Carica nel registro B il contenuto della locazione di memoria posta all'indirizzo specificato
<b>Istruzioni di trasferimento da un registro alla memoria</b>					
3	<b>mov [indirizzo],A</b>	0001 1001 nnnnnnnn (indirizzo)	19	25	Carica il contenuto del registro A nella locazione di memoria posta all'indirizzo specificato
4	<b>mov &lt;var&gt;,B</b>	0001 1010 nnnnnnnn (indirizzo)	1A	26	Carica il contenuto del registro B nella locazione di memoria posta all'indirizzo specificato
<b>Istruzioni di trasferimento tra registri</b>					
5	<b>mov B,A</b>	0001 0010	12	18	Carica il contenuto del registro A nel registro B
6	<b>mov A,B</b>	0001 0001	11	17	Carica il contenuto del registro B nel registro A
<b>Istruzioni di trasferimento di un valore immediato in un registro</b>					
7	<b>mov A,n</b>	0001 0110 nnnnnnnn (valore)	16	22	Carica nel registro A il valore n
8	<b>mov B,n</b>	0001 0111 nnnnnnnn (valore)	17	23	Carica nel registro B il valore n
<b>Istruzioni aritmetico-logiche</b>					
9	<b>add A,B</b>	0010 0001	21	33	$A \leftarrow A + B$ Somma al contenuto del registro A il contenuto del registro B e pone il risultato nel registro A
10	<b>add B,A</b>	0010 0010	22	34	$B \leftarrow B + A$ Somma al contenuto del registro B il contenuto del registro A e pone il risultato nel registro B
11	<b>add A,n</b>	00100011 nnnnnnnn (valore)	23	35	$A \leftarrow A + nnnnnnnn$ Somma al contenuto del registro A il valore n e pone il risultato nel registro A
12	<b>sub A,B</b>	0011 0001	31	49	$A \leftarrow A - B$ Sottrae al contenuto del registro A il contenuto del registro B e pone il risultato nel registro A
13	<b>inc A</b>	0100 0001	41	97	$A \leftarrow A + 1$ Incrementa di 1 il valore di A
14	<b>inc B</b>	0101 0010	52	82	$B \leftarrow B + 1$ Incrementa di 1 il valore di B
15	<b>dec A</b>	0110 0001	61	97	$A \leftarrow A - 1$ Decrementa di 1 il valore di A
16	<b>dec B</b>	0111 0010	72	144	$B \leftarrow B - 1$ Decrementa di 1 il valore di B
17	<b>cmp A,n</b>	0011 0010 nnnnnnnn (valore)	32	50	Confronta (compare) il valore contenuto nel registro A con il valore n. È una sottrazione $A - n$

#	Istruzione Assembly - mnemonica	Codice Operativo (codice macchina binario)	hex	dec	Descrizione
18	<b>not A</b>	0011 0100	34	52	Pone nel registro A il valore di A negato
19	<b>or A,B</b>	0011 0101	35	53	Effettua l'OR logico bit a bit tra il contenuto del registro A e il contenuto del registro B e pone il risultato in A
20	<b>and A,B</b>	0011 0110	36	54	Effettua l'AND logico bit a bit tra il contenuto del registro A e il contenuto del registro B e pone il risultato in A
<b>Istruzioni di salto</b>					
21	<b>jnz &lt;indirizzo/etichetta di salto&gt;</b>	1001 0010 nnnnnnnn (et/ind)	92	146	Salto condizionato: salta all'indirizzo di memoria indicato in <indirizzo/etichetta> se il Flag Zero è a zero, cioè se è NON-Zero (NZ)
22	<b>jz &lt;indirizzo/etichetta di salto&gt;</b>	1001 0001 nnnnnnnn (et/ind)	91	145	Salto condizionato: salta all'indirizzo di memoria indicato in <indirizzo/etichetta> se il Flag Zero è a 1, cioè se è Zero (Z)
23	<b>jmp &lt;indirizzo/etichetta di salto&gt;</b>	1001 0000 nnnnnnnn (et/ind)	90	144	Salto non condizionato: salta all'indirizzo di memoria indicato in <indirizzo/etichetta>
<b>Istruzioni che coinvolgono lo stack</b>					
24	<b>call &lt;indirizzo/etichetta di salto&gt;</b>	1001 1000 nnnnnnnn (et/ind)	98	152	Chiamata di routine: salva il PC nello stack tramite push e salta all'indirizzo di memoria indicato in <indirizzo/etichetta>
25	<b>ret</b>	1001 1001	99	153	Ripristina il PC, prelevandolo dallo stack tramite pop
26	<b>push A</b>	1001 0000	A0	160	Carica il contenuto del registro A sulla cima dello stack (locazione puntata dallo SP) e decremente SP (SP = SP - 1)
27	<b>push B</b>	1001 0001	A1	161	Carica il contenuto del registro B sulla cima dello stack (locazione puntata dallo SP) e decremente SP (SP = SP - 1)
28	<b>pop A</b>	1000 0000	80	128	Incrementa SP (SP = SP + 1) e carica il contenuto della cima dello stack (locazione puntata da SP) nel registro A
29	<b>pop B</b>	1000 0001	81	129	Incrementa SP (SP = SP + 1) e carica il contenuto della cima dello stack (locazione puntata da SP) nel registro B
<b>Istruzioni di Ingresso / Uscita (Input/Output)</b>					
30	<b>in A, &lt;indirizzo unità ingresso&gt;</b>	0000 1000 nnnnnnnn (indirizzo)	08	08	Carica nel registro A il contenuto della periferica posta all'indirizzo <indirizzo unità di ingresso> (parametro)
31	<b>out A, &lt;indirizzo unità&gt;</b>	0000 1001 nnnnnnnn (indirizzo)	09	09	Carica nella periferica posta all'indirizzo <indirizzo unità di uscita> il contenuto del registro A
<b>Altre istruzioni</b>					
32	<b>nop</b>	0000 0000	00	00	Nessuna operazione
33	<b>halt</b>	1111 1111	FF	255	Ferma il programma

## 8 Le categorie di istruzioni del linguaggio Assembly DuplOne

### Istruzioni di trasferimento

Sono le istruzioni con cui si effettuano le operazioni di:

- **lettura da memoria.** Esempio: `mov B, [02h]` è una lettura dalla memoria, poiché l'operando sorgente è un indirizzo di memoria e il destinatario è un registro della CPU, quindi il trasferimento va dalla RAM alla CPU;
- **scrittura in memoria.** Esempio: `mov [02h], B` è una scrittura in memoria, poiché l'operando sorgente è un registro della CPU e il destinatario è un indirizzo di memoria, quindi il trasferimento va dalla CPU alla RAM;
- **lettura da periferica.** Esempio: `in A, 2` è lettura dalla periferica di indirizzo 2;
- **scrittura su periferica.** Esempio: `out A, 2` è una scrittura nella periferica di indirizzo 2;
- **trasferimento tra registri;**
- **trasferimento di un valore in un registro.**

### Istruzioni aritmetico-logiche

Effettuano somme, sottrazioni, incremento, decremento e operazioni logiche (OR, AND, NOT) e modificano il registro di stato in base al risultato:

- Z = 1: il flag di Zero è Vero, se il risultato dell'ultima istruzione aritmetico/logica è uguale a zero;
- Z = 0: il flag di Zero è Falso, se il risultato dell'ultima istruzione aritmetico/logica è diverso da zero;
- S = 1: il flag di Segno è Vero, se il risultato dell'ultima istruzione aritmetico/logica è negativo;
- S = 0: il flag di Segno è Falso, se il risultato dell'ultima istruzione aritmetico/logica non è negativo.

Un'osservazione importante riguarda l'istruzione *compare* (`cmp`), che confronta due operandi.

Confrontare «*a*» con «*b*» significa stabilire una relazione d'ordine tra loro. Questo può essere fatto osservando il risultato della loro differenza *a - b*:

se <i>a - b</i> è negativo	<i>a &lt; b</i>
se <i>a - b</i> è positivo	<i>a &gt; b</i>
se <i>a - b</i> è uguale a zero	<i>a = b</i>

L'istruzione `cmp` effettua una sottrazione (per questo è classificata come istruzione aritmetica) e, in base al risultato, **forza il flag di Zero (Z) e il flag di Segno (S) presenti nel registro di stato senza variare il contenuto dei registri.**

Successivamente, con una istruzione di salto, i flag vengono testati per trarre le conclusioni del confronto.

Il confronto usato nelle condizioni con le strutture di controllo in un linguaggio ad alto livello, in Assembly è risolto utilizzando due istruzioni: *compare* e *jump*.

### Istruzioni di salto

Le istruzioni di salto (`jmp`) sono utilizzate in Assembly per codificare le strutture di controllo tipiche di un linguaggio di programmazione ad alto livello, come IF...THEN...ELSE, WHILE...

«Eseguire il salto» significa che la CPU «forza» nel Program Counter l'indirizzo che trova come operando. Ci sono due tipi di istruzioni di salto:

- **istruzioni di salto condizionato:** eseguono un test su uno o più bit del registro di stato; se l'esito del test è positivo «eseguono un salto» all'istruzione il cui indirizzo è associato a <etichetta>:
  - **jz** equivale a JUMP IF Zero; il programma «salta» se il *Flag Z* è Vero, cioè uguale a 1; Z è Vero se il risultato dell'ultima operazione aritmetica è uguale a zero;
  - **jnz** equivale a JUMP IF NOT Zero: il programma «salta» se il *Flag Z* = 0, cioè NOT ZERO;
- **istruzioni di salto non condizionato:** eseguono un salto all'istruzione il cui indirizzo è associato a <etichetta> senza effettuare alcun test:
  - **jmp** <etichetta>.

### Pit Stop

10 Che differenza c'è tra le istruzioni di salto condizionato e quelle di salto non condizionato?

### Esempio 3

Consideriamo il seguente frammento di programma che effettua un conteggio. Il programma contiene un'istruzione di salto condizionato (**jnz**).

<b>Start:</b>	mov B,0 mov A,5
<b>Ripeti:</b>	inc B dec A jnz <b>Ripeti</b> mov [50h],B

Vediamo ora nei dettagli come avviene l'esecuzione del programma.

Indirizzo di memoria	Codice macchina	Label	Istruzioni Assembly	Note
00010000	00010111	Start	mov B,0	Trasferisce il valore 0 nel registro B.
00010001	00000000		mov A,5	Trasferisce il valore 5 nel registro A.
00010010	00010110		inc B	Incrementa il contenuto del registro B. È un'istruzione aritmetica e quindi modifica il registro di stato, almeno nel <i>Flag Zero</i> .
00010011	00000101		dec A	Decrementa il contenuto del registro A. È un'istruzione aritmetica che modifica <i>Flag Zero</i> presente nel registro di stato.
00010100	01010010	Ripeti	jnz Ripeti	È un'istruzione di salto condizionato che ha come operando l'etichetta «Ripeti», un nome generico usato per individuare l'indirizzo logico di memoria a cui il programma deve saltare.
00010110	10010010			
00010111	00010100			
00011000	00011010			
00011001	01010000			

Di seguito focalizziamo l'attenzione sull'esecuzione dell'istruzione di salto (per semplicità è omessa la fase di decode):

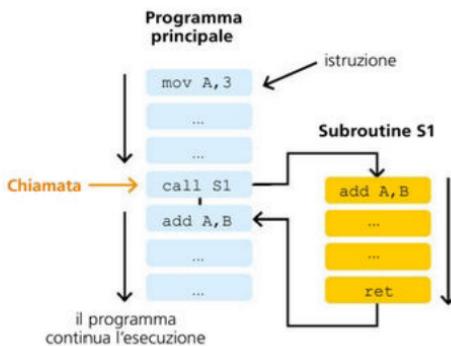
- allo stato iniziale PC = 00010000;
- dopo aver fatto la fetch di jnz Ripeti, il PC viene aggiornato e punta già all'istruzione successiva. Infatti contiene PC = 00011000, che è l'indirizzo dell'istruzione mov [50h], B;
- in fase di esecuzione della jnz, tuttavia, se l'esito del test è positivo, nel PC viene messo l'indirizzo corrispondente all'operando. Nel nostro esempio l'operando è **Ripeti**, che corrisponde all'indirizzo 00010100 (che contiene l'istruzione inc B). L'effetto di questa «forzatura» è che la prossima istruzione che andrà in esecuzione non sarà più la successiva a jnz, ma quella puntata dal nuovo valore del PC: la CPU ha fatto un salto avanti, se l'indirizzo è maggiore di quello dell'istruzione successiva, indietro se è minore (come nel nostro esempio). Osserviamo che il valore del PC viene modificato prima della execute; questo significa che prima di eseguire l'istruzione corrente il Program Counter punta già all'istruzione successiva. La conseguenza è che viene ripetuta l'esecuzione di alcune istruzioni; viene cioè implementata la **struttura di controllo ciclo**. Quando il test sarà Falso, allora non verrà forzato il PC e l'esecuzione proseguirà passando all'istruzione successiva (uscita dal ciclo).

#### ■ notabene

Una routine o subroutine (chiamata anche *funzione*, *procedura* o *sottoprogramma*) è una porzione di codice che può essere chiamata più volte in un programma per migliorarne l'efficienza. Per esempio, una routine che restituisce il perimetro di un quadrato può essere chiamata più volte applicandola a numeri differenti.

#### CALL

L'istruzione call è una particolare istruzione di salto. La sintassi è <call indirizzo/etichetta>, simile a un normale salto non condizionato della forma <jmp indirizzo/etichetta>, ma con una grande differenza: l'istruzione call è utilizzata per «saltare» a una parte di codice (routine), eseguito fino a quando si incontra un'istruzione ret (RETURN, ritorno), che fa ritornare all'istruzione successiva alla call (**Figura 8**).



→ Fig. 8 La chiamata di una subroutine.

#### Esempio 4

Consideriamo lo stato del sistema in un determinato istante, rappresentato in **Tavella 3**. Per semplicità indichiamo con *Istruzione* *n* una generica istruzione che può comunque avere lunghezza variabile. Supponiamo, come mostrato nella tabella, che il programma principale sia allocato dalla locazione 00001000 e che la routine sia allocata dalla locazione 00110011 («etichettata» con *ippo*) fino alla locazione 00110110.

Programma principale		Routine	
Indirizzo	Istruzione	Label	Indirizzo
...	...		
00001000	Istruzione 1	pippo	00110011
00001001	Istruzione 2		00110100
00001010	Istruzione 3		00110101
00001011	Istruzione 4		00110110
00001100	call pippo		ret
00001101	Istruzione 5		
00001110	...		

Tab. 3 Esecuzione della call.

La CPU, per eseguire il programma, si comporta nel modo seguente:

...

#### Istruzione 4

- FETCH di istruzione 4;
- aggiornamento del contenuto del PC con l'indirizzo della prossima istruzione (PC = 00001100);
- EXECUTE di istruzione 4.

#### Istruzione CALL

- FETCH di call pippo;
- la CPU carica PC con 00001101, che è l'indirizzo di istruzione 5;
- EXECUTE di call pippo: il PC viene forzato con l'indirizzo di «pippo» (00110011) in modo che la prossima istruzione che sarà eseguita corrisponderà a istruzione A.

#### Istruzioni della routine che è stata chiamata

- FETCH di istruzione A;
- PC = 00110100;
- EXECUTE di istruzione A;
- FETCH di istruzione B;
- PC = 00110101;
- EXECUTE di istruzione B;
- FETCH di istruzione C;
- PC = 00110110;
- EXECUTE di istruzione C;
- FETCH di istruzione ret;
- EXECUTE di ret: il PC è forzato con il valore dell'indirizzo dell'istruzione 5.  
Eseguire la ret significa forzare il PC con l'indirizzo dell'istruzione 5 e tornare al programma principale.

#### Ritorno al programma principale

- FETCH di istruzione 5;
- ...

**Come fa la CPU a sapere qual è l'indirizzo dell'istruzione successiva alla call?**

Qui sta la vera differenza tra un'istruzione di salto e la `call`: l'istruzione di salto non ritorna al programma principale, quindi non ha necessità di conoscere l'indirizzo di «rientro», al contrario dell'istruzione `call` che ha bisogno, per così dire, di «rientrare alla base».

Per non perdere l'informazione dell'indirizzo di ritorno la CPU, prima di forzare il PC con il nuovo indirizzo, deve salvare il PC nello stack.

La sequenza corretta di esecuzione diventa (**Tabella 4**):

**Istruzione CALL**

- FETCH di `call pippo;`
- PC = 00001101;
- EXECUTE di `call pippo:` push 00001101 (salva sulla cima dello stack), PC = 00110011 con salto all'istruzione A.

**Istruzioni della routine che è stata chiamata**

- FETCH di istruzione A;
- ...;
- FETCH `ret`:
- PC = 00110111 (indirizzo dell'istruzione successiva alla `ret`);
- pop 00001101, che preleva dallo stack il dato che è in cima, quindi quello messo dall'ultima push: l'indirizzo di ritorno;
- PC = 00001101 (forza il PC con l'indirizzo dell'istruzione 5).

**Ritorno al programma principale**

- FETCH di istruzione 5;
- ...

Istruzione	Operazioni della CPU
<code>call</code>	<ul style="list-style-type: none"> <li>• Salva nello stack l'indirizzo dell'istruzione successiva;</li> <li>• forza il PC con l'indirizzo specificato nell'operando.</li> </ul>
<code>ret</code>	<ul style="list-style-type: none"> <li>• Preleva dallo stack l'ultimo dato immesso;</li> <li>• forza il PC con il dato prelevato dallo stack;</li> <li>• il dato prelevato dallo stack durante l'esecuzione della <code>ret</code> è l'indirizzo di ritorno al programma principale.</li> </ul>

↑ Tab. 4 Operazioni di `call` e `ret`.

## 9 Le strutture di controllo

Le strutture di controllo sono usate nei linguaggi ad alto livello per controllare il flusso di esecuzione di un programma.

Tutti i linguaggi ad alto livello strutturati sono dotati di strutture di controllo che si dividono in:

- selezione (`If...the...else, switch, case`);
- iterazione (`for, while, do while`).

Entrambe queste strutture utilizzano una condizione per «decidere» quale sarà la prossima istruzione da eseguire, controllando in questo modo il flusso di esecuzione e facendo perdere la sequenzialità.

Per esempio:

- nella struttura di selezione *if...then...else*, se la condizione dopo *if* è vera, vengono eseguite le istruzioni che vengono dopo *then*, altrimenti quelle che vengono dopo *else*;
- nella struttura di iterazione, se la condizione è vera, vengono ripetute delle istruzioni (ciclo), se invece la condizione è falsa, il programma prosegue in modo sequenziale.

Il linguaggio Assembly **non è dotato di strutture di controllo**.

Per tradurre una struttura di controllo in Assembly si utilizzano le istruzioni di salto, che testano i flag del registro di stato. L'operando dell'istruzione è un indirizzo di memoria che sarà assegnato al Program Counter:

- per la selezione il salto viene fatto in avanti, a un indirizzo successivo. In questo modo non vengono eseguite alcune istruzioni;
- per l'iterazione il salto viene fatto indietro, a un indirizzo precedente. In questo modo si ripetono alcune istruzioni.

## 10 Metodi di indirizzamento

Per metodo di indirizzamento si intende **il modo in cui la CPU reperisce gli operandi**.

### ■ Metodo di indirizzamento a registro

Gli operandi sono i registri.

L'indirizzamento a registro implica che **gli operandi sono codificati nel codice operativo stesso**. Per esempio:

```
inc A  è codificata con  01000001
      inc B  è codificata con  01010010
```

### Esempio 5

```
add A,B
mov A,B
inc B
```

### ■ Metodo di indirizzamento immediato

**L'operando** (sorgente) è parte dell'istruzione.

Ad esempio, l'istruzione *mov B, 7* è codificata con 2 byte: il primo è il codice operativo e il successivo byte è il dato.

### Esempio 6

```
mov B,7  dove 7 è l'operando sorgente
Codice macchina: 00010111  00000111
```

### ■ Metodo di indirizzamento diretto

L'operando è contenuto in una locazione di memoria di cui, nell'istruzione, è esplicitato l'indirizzo (**Figura 9**).

La dimensione dell'istruzione è tale da dover contenere il codice operativo e l'indirizzo di memoria.

#### Esempio 7

`mov A, [12h]` che si legge: «trasferisci il contenuto della locazione di indirizzo 12h (operando sorgente) nel registro A».

Il tempo necessario per eseguire un'istruzione a indirizzamento diretto è maggiore rispetto a quello immediato, poiché la CPU, dopo aver fatto la fetch del codice operativo e dopo aver caricato l'indirizzo della locazione, deve fare un ulteriore accesso alla memoria per prelevare il dato.

Questo tipo di indirizzamento coinvolge operazioni sia di lettura sia di scrittura.



→ **Fig. 9** Indirizzamento diretto. L'indirizzo dell'operando è contenuto nell'istruzione.

### ■ Metodo di indirizzamento indiretto tramite registro

L'operando è contenuto in una locazione di memoria il cui indirizzo è inserito in un registro: l'istruzione specifica il registro che contiene l'indirizzo dell'operando (**Figura 10**).

#### Esempio 8

`mov [B], A` che si legge: «trasferisci il contenuto del registro A nella locazione di memoria il cui indirizzo è nel registro B».

Se, per esempio, B=12h, allora `mov [B], A` e `mov [12h], A` hanno lo stesso effetto.



→ **Fig. 10** Indirizzamento indiretto tramite registro: l'indirizzo dell'operando è contenuto in una locazione di memoria il cui valore è contenuto nel registro specificato dall'istruzione.

# Approfondimento - Tecnologia

## La programmazione dei sistemi di Intelligenza Artificiale e l'importanza dei dati che mandiamo in rete

ORIENTAMENTO  
**STEM**

Un'importante osservazione riguarda il modo di programmare i sistemi di Intelligenza Artificiale.

Se la programmazione tradizionale applica lo schema «if-then-else» (ad esempio: «SE piove ALLORA apri l'ombrelllo, ALTRIMENTI guarda il cielo!»), in cui si prevedono a priori tutte le possibilità, la programmazione dei sistemi di Intelligenza Artificiale cerca di addestrare la macchina a rispondere in maniera autonoma a un problema imparando dall'esperienza. In pratica gli algoritmi «adattivi» di *machine learning* apprendono direttamente dai dati che vengono loro forniti: tanti più sono i dati disponibili, tanto migliori sono i risultati che ottengono.

Ecco perché i nostri dati che mandiamo in rete (quello che siamo, facciamo, vediamo, scriviamo, filmiamo) sono così importanti per chi produce software. Nessuno ci regala niente, anzi, siamo noi, con le nostre azioni, che forniamo il carburante per i motori di Intelligenza Artificiale o, in altri termini, «se non stai pagando per un prodotto, allora il prodotto sei TU».



# Laboratorio

## DuplOne Assembly Simulator

DuplOne è un microprocessore virtuale ideato dagli Autori del corso; possiede un set di istruzioni ridotto, ma efficace, molto utile per apprendere i rudimenti della programmazione Assembly.

La sua architettura è spiegata nel Paragrafo 6.

Per eseguire un programma scritto in Assembly DuplOne, utilizziamo DuplOne Assembly Simulator, un ambiente virtuale che simula l'esecuzione di un programma scritto con le istruzioni Assembly di DuplOne e fornisce tutti gli strumenti didattici necessari per il caricamento di un file, la sua esecuzione, il debug e il controllo dei registri e della memoria. È utilizzabile via Web, da dispositivi fissi e mobili.

### ESEMPIO di utilizzo

- Scrivere il codice sorgente in Assembly DuplOne utilizzando un qualsiasi editor di testi e salvarlo con una delle estensioni .asm, .dsm, .txt.

Supponiamo di scrivere il seguente codice:

```
mov a,0b00010000      ;binario
mov b, 0x20            ;resadecimale
add a,b
mov a,3                ;decimale
mov b,0b11111111      ;-1 in complemento a 2
```



DuplOne  
Assembly  
Simulator

- Per utilizzare il simulatore, andare all'indirizzo <http://u.deascuola.it/duplone>. L'interfaccia del simulatore si presenta come è mostrato in Figura 1.

DuplOne Assembly Simulator		
Load	Reload	Reset
Ind	Val	Binario
00:	22	00010110
01:	16	00010000
02:	23	00010111
03:	32	00100000
04:	33	00100001
05:	22	00010110
06:	3	00000001
07:	23	00010111
08:	-1	11111111

Istruzioni		
00:	MOV A, constant	
01:	16	
02:	MOV B, constant	
03:	32	
04:	ADD A, B	
05:	MOV A, constant	
06:	3	
07:	MOV B, constant	
08:	-1	

Scatta	Ptr	Memory Dump
A	0	00000000
B	0	00000000
PC	0	00000000
SP	255	11111111
SR	0	0
	ZERO	CARRY

Fig. 1

- La tabella seguente sintetizza i comandi disponibili e le loro funzioni:

<b>Load</b>	Utilizzato per caricare il file sorgente. Se non ci sono errori, il programma è caricato e sono visualizzabili le istruzioni, il contenuto della memoria (indirizzo con il contenuto mostrato in decimale e binario) e il contenuto dei registri (in decimale e binario).
<b>Reload</b>	Ricarica dalla memoria il programma precedente.
<b>Reset</b>	Ripulisce memoria e registri, azzerandone il contenuto.
<b>Step back</b>	Esegue le istruzioni, passo passo, a ritroso, aggiornando il contenuto di memoria e registri.
<b>Single step</b>	Esegue le istruzioni, passo passo, aggiornando il contenuto di memoria e registri.
<b>Run</b>	Esegue il programma, eseguendo le istruzioni, una dopo l'altra, aggiornando il contenuto di memoria e registri.
<b>Scarica Pdf</b>	Scarica il manuale Assembly DuplOne in formato PDF.
<b>Memory Dump</b>	Mostra il contenuto della memoria in decimale e binario.

Note per l'utilizzo e la scrittura del codice:

- i dati in memoria sono raggiungibili solo tramite indirizzo fisico (cioè non si possono dichiarare variabili con un nome);
- le etichette non sono previste (nella versione 1.0). I «salti» di programma sono possibili solo con indirizzi fisici;
- i numeri esadecimali si possono inserire nel programma facendoli precedere dal prefisso «0x»; ad esempio, 0x0F indica il valore esadecimale 0F (15 decimale);
- si possono inserire nel programma numeri binari facendoli precedere da «0b». Ad esempio, 0b00010000 indica il valore decimale 32;
- I commenti sono preceduti da «;».

## Programmare con il linguaggio Assembly DuplOne

### ATTIVITÀ 1

Tradurre nel linguaggio Assembly DuplOne istruzioni scritte nel linguaggio naturale

#### Scenario

Si dispone delle seguenti istruzioni scritte in linguaggio naturale:

- trasferisci nel registro A il contenuto della locazione di memoria di indirizzo 34h
- trasferisci nel registro B il contenuto della locazione di memoria di indirizzo 34h
- trasferisci nella locazione di memoria di indirizzo 56h il contenuto del registro A
- trasferisci nella locazione di memoria di indirizzo 75h il contenuto del registro B
- somma il contenuto del registro A al contenuto del registro B e poni il risultato in A

- f. sottrai il contenuto del registro B al contenuto del registro A e poni il risultato in A
- g. incrementa il contenuto del registro A
- h. salta alla locazione di indirizzo 08h, se il risultato dell'ultima istruzione aritmetico/logica è zero
- i. esegui la routine posta all'indirizzo A8h
- j. trasferisci il valore 34h nel registro A

### Consegna

1. Scrivere le istruzioni in linguaggio Assembly DuplOne.
2. Tradurre le istruzioni Assembly in binario, specificando per ogni istruzione:
  - la lunghezza in byte;
  - il metodo di indirizzamento;
  - la categoria di appartenenza al set di istruzioni.

### Svolgimento

Per rispondere alle domande occorre consultare il set di istruzioni Assembly DuplOne.

#### a) trasferisci nel registro A il contenuto della locazione di memoria di indirizzo 34h

<b>1. Codice Assembly</b>	mov A, [34h]
<b>2. Codice binario</b>	00010011 00110100
• lunghezza istruzione	2 byte
• metodo di indirizzamento	diretto
• categoria	trasferimento

#### b) trasferisci nel registro B il contenuto della locazione di memoria di indirizzo 34h

<b>1. Codice Assembly</b>	mov B, [34h]
<b>2. Codice binario</b>	00011110 00110100
• lunghezza istruzione	2 byte
• metodo di indirizzamento	diretto
• categoria	trasferimento

#### c) trasferisci nella locazione di memoria di indirizzo 56h il contenuto del registro A

<b>1. Codice Assembly</b>	mov [56h], A
<b>2. Codice binario</b>	00011001 01010110
• lunghezza istruzione	2 byte
• metodo di indirizzamento	diretto
• categoria	trasferimento

**d) trasferisci nella locazione di memoria di indirizzo 75h il contenuto del registro B**

<b>1. Codice Assembly</b>	mov [75h], B
<b>2. Codice binario</b>	00011010 01110101
• lunghezza istruzione	2 byte
• metodo di indirizzamento	diretto
• categoria	trasferimento

**e) somma il contenuto del registro A al contenuto del registro B e poni il risultato in A**

<b>1. Codice Assembly</b>	add A, B
<b>2. Codice binario</b>	00100001
• lunghezza istruzione	1 byte
• metodo di indirizzamento	a registro
• categoria	aritmetico/logica

**f) sottrai il contenuto del registro B al contenuto del registro A e poni il risultato in A**

<b>1. Codice Assembly</b>	sub A, B
<b>2. Codice binario</b>	00110001
• lunghezza istruzione	1 byte
• metodo di indirizzamento	a registro
• categoria	aritmetico/logica

**g) incrementa il contenuto del registro A**

<b>1. Codice Assembly</b>	inc A
<b>2. Codice binario</b>	01000001
• lunghezza istruzione	1 byte
• metodo di indirizzamento	a registro
• categoria	aritmetico/logica

**h) salta alla locazione di indirizzo 08h se il risultato dell'ultima istruzione aritmetico/logica è zero**

<b>1. Codice Assembly</b>	jz 08h
<b>2. Codice binario</b>	10010001 00001000
• lunghezza istruzione	2 byte
• metodo di indirizzamento	diretto
• categoria	salto

**i) esegui la routine posta all'indirizzo 08h**

<b>1. Codice Assembly</b>	call 08h
<b>2. Codice binario</b>	10011000 00001000
• lunghezza istruzione	2 byte
• metodo di indirizzamento	diretto
• categoria	istruzione sullo stack (salto con ritorno)

**l) trasferisci il valore 34h nel registro A**

<b>1. Codice Assembly</b>	mov A,34h
<b>2. Codice binario</b>	00010110 00110100
• lunghezza istruzione	2 byte
• metodo di indirizzamento	immediato
• categoria	trasferimento

**ATTIVITÀ 2****Utilizzo del linguaggio Assembly DuplOne****Scenario**

Eseguire un *countdown* a partire da un numero letto da input dalla periferica 5.

**Consegna**

1. Scrivere un programma nel linguaggio Assembly DuplOne che esegua il compito.
2. Scrivere in memoria (locazione 44h) il numero di iterazioni che sono state fatte.

**Svolgimento****Punto 1**

```

in A, 5    legge un dato dalla periferica 5 di input
mov B, 0
ciclo:
    inc B
    dec A
    jnz ciclo
  
```

**Punto 2**

```
    mov [44h], B
```

**ATTIVITÀ 3****Dalla pseudo codifica al linguaggio macchina****Scenario**

Si dispone del seguente algoritmo, scritto con uno pseudocodice.

```

leggi a
leggi b
if ( a == 4 ) then
    a++;
else
    b++;
fine if

```

### Consegna

1. Codificare l'algoritmo in linguaggio Assembly DuplOne.
2. Successivamente, tradurre il programma ottenuto in linguaggio macchina DuplOne.

### Svolgimento

#### Punto 1

1. Si supponga che la *symbol table*, che associa il nome di una variabile a un indirizzo di memoria, sia:

SYMBOL TABLE	
Name	Address
a	24h
b	25h

Quindi la variabile *a* è associata all'indirizzo di memoria 24h, la variabile *b* all'indirizzo 25h.

2. Codifica in linguaggio Assembly DuplOne:

```

in A,5
mov [24h], A
in A,5
mov [25h], A
mov A, [24h]
mov B, [25h]
cmp A, 04h
jnz salto
inc A
jmp salto2
salto: inc B
salto2: mov [24h], A
        mov [25h], B
fine:

```

	in A, 5	leggi il dato dalla periferica 5 e trasferiscilo in A
	mov [24h], A	trasferisci nella locazione di indirizzo 24h il valore del registro A. Ora la locazione 24h contiene il primo valore letto da input. È stato assegnato un valore alla variabile a
	in A, 5	leggi il dato dalla periferica 5 e trasferiscilo in A
	mov [25h], A	trasferisci nella locazione di indirizzo 25h il valore del registro A. Ora la locazione 25h contiene il secondo valore letto da input. È stato assegnato un valore alla variabile b
	mov A, [24h]	lettura da memoria: trasferisci il contenuto della locazione di indirizzo 24h nel registro A
	mov B, [25h]	lettura da memoria: trasferisci il contenuto della locazione di indirizzo 25h nel registro B
	cmp A, 04h	sottrai 4 al contenuto di A
	jnz salto	se il flag di Zero non è vero, cioè se FZ = 0, salta all'indirizzo associato all'etichetta «salto»
	inc A	incrementa il valore del registro A
	jmp salto2	salta senza condizioni all'indirizzo associato all'etichetta salto2
salto:	inc B	incrementa il valore del registro B
salto2:	mov [24h], A	scrittura in memoria: trasferisci il contenuto del registro A nella locazione di memoria 24h
	mov [25h], B	trasferisci il contenuto del registro B nella locazione 25h
	fine:	

**Punto 2**

Supponiamo che il primo valore letto sia 3 e il secondo sia 2 e che il programma sia allocato a partire dall'indirizzo 00h. In tal caso, la traduzione in codice macchina è la seguente.

Indirizzi di memoria	Istruzioni
00000000	00001000
00000001	00000101
00000010	00011001
00000011	00011000
00000100	00000100
00000101	00000101
00000110	00011001
00000111	00011001
00001000	00010011
00001001	00011000
00001010	00011110
00001011	00011001
00001100	00110010
00001101	00000100
00001110	10010010
00001111	01010010



00010000	01000001
00010001	01010010
00010010	00011001
00010011	00011000
00010100	00011010
00010101	00011001
00010110	.....
00010111	.....
00011000 variabile a	00000011
00011001 variabile b	00000010 variabile b

## ATTIVITÀ 4

### Utilizzo dei flag del registro di stato

#### Scenario

Dato il seguente programma scritto in linguaggio Assembly DuplOne:

```
mov A, 07h
mov B, 09h
inc A
inc A
sub A, B
```

#### Consegna

1. Stabilire quanto valgono al termine del programma FZ e FS e perché.
2. Tradurre il programma in linguaggio macchina DuplOne.

#### Svolgimento

##### Punto 1

L'ultima istruzione aritmetica che modifica il registro di stato è sub A,B.

In questo caso effettua la sottrazione **9 - 9 = 0**. Pertanto **FZ = 1** (risultato zero) e **FS = 0** (risultato non negativo).

##### Punto 2

mov A, 07h	00010110 00000011
mov B, 09h	00011110 00001001
inc A	01000001
inc A	01000001
sub A, B	00110001

**ATTIVITÀ 5****Dal linguaggio ad alto livello al linguaggio Assembly, fino al linguaggio macchina****Scenario**

Dato il seguente programma scritto in linguaggio ad alto livello (in questo caso Java):

```
int pippo=0;
int i;
for (i=0; i<5; i++) {
    pippo +=2;
}
```

**Consegna**

1. Scrivere il programma in linguaggio Assembly DuplOne.
2. Tradurre il programma ottenuto in linguaggio macchina.

**Svolgimento***Punto 1*

È data la tabella seguente:

SYMBOL TABLE	
Name	Address
pippo	80h
ciclo	88h

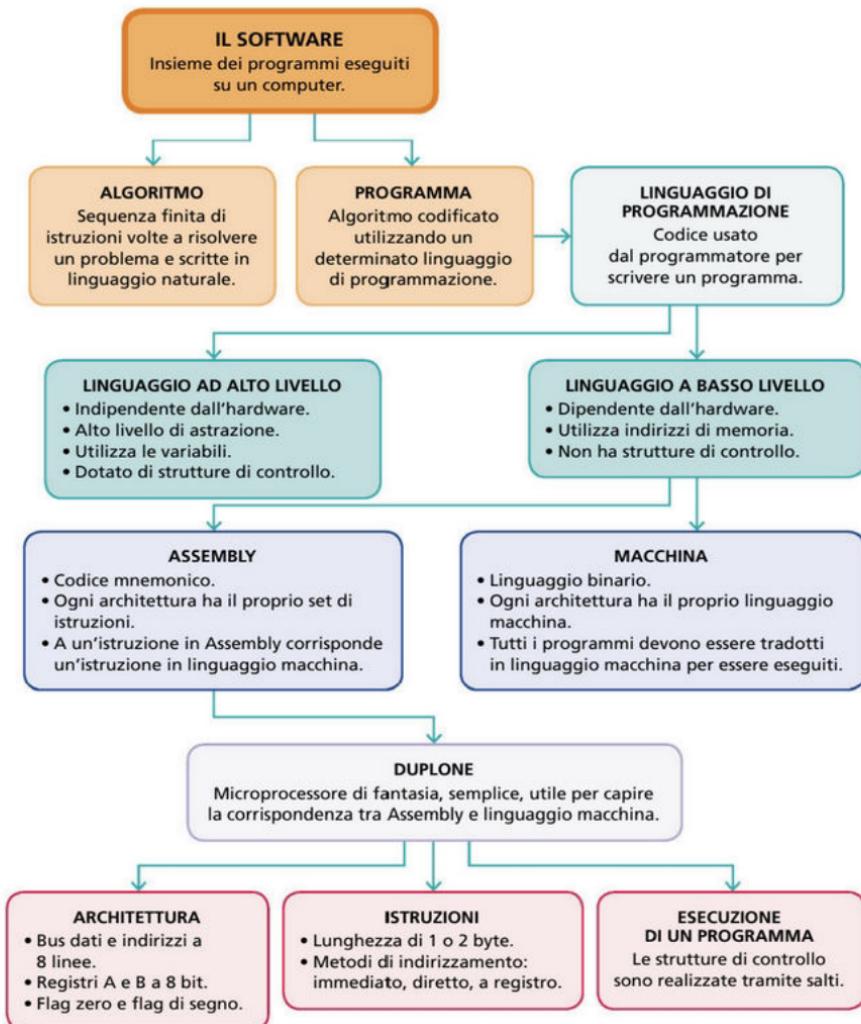
```
        mov B, 0
        mov A, 0
ciclo: inc B
        inc B
        inc A
        cmp A, 5
        jnz ciclo

        mov [80h], B
```

*Punto 2*

mov B, 0	00011110 00000000
mov A, 0	00010011 00000000
inc B	01010010
inc B	01010010
inc A	01000001
cmp A, 5	00110010 00000101
jnz ciclo	10010010 10001000
mov [80h], B	00011010 10000000

# SINTESI



# Verifica delle CONOSCENZE

Unità 4



## Vero o Falso?

- 1 Un algoritmo è un programma.
- 2 La CPU legge dalla RAM degli algoritmi e li esegue.
- 3 Il C è un linguaggio di programmazione di alto livello.
- 4 L'Assembly è il linguaggio macchina.
- 5 Il codice sorgente è caricato nella RAM per essere eseguito.
- 6 Un linguaggio di basso livello ha un basso livello di astrazione.
- 7 Un linguaggio di basso livello dipende dall'hardware.
- 8 Se un linguaggio è di basso livello, allora è più semplice.
- 9 Nel metodo di indirizzamento immediato il dato è contenuto nell'istruzione.
- 10 La CPU è in grado di interpretare qualsiasi algoritmo.

V  F

V  F

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.

- 11 Una generica istruzione Assembly:
- a A ha sempre due operandi
- b è costituita da codice operativo e operandi
- c è sempre lunga 2 byte
- d ha solo operandi
- 12 In una istruzione Assembly:
- a il codice operativo indica l'operazione che la CPU dovrà eseguire
- b il codice operativo segue gli operandi
- c l'operando sorgente è sempre il primo
- d l'operando destinazione è sempre il primo
- 13 Una cella di memoria RAM:
- a è una variabile
- b contiene un indirizzo

- c è nella CPU
- d contiene sempre un'istruzione

- 14 L'istruzione call:
- a equivale a una jump
- b effettua un salto senza ritorno
- c opera sullo stack
- d è una push
- 15 Le strutture di controllo di un linguaggio ad alto livello:
- a sono presenti anche in Assembly
- b non possono essere implementate in Assembly
- c possono essere implementate in Assembly tramite la call
- d nessuna delle precedenti risposte è vera
- 16 La dimensione di un'istruzione macchina:
- a è sempre fissa
- b è sempre di due byte
- c dipende dalla velocità della cpu
- d nessuna delle precedenti
- 17 L'istruzione CMP (compare) è un'istruzione:
- a logica
- b di salto
- c aritmetica
- d di trasferimento
- 18 In Assembly un'etichetta corrisponde a:
- a un indirizzo di memoria
- b una variabile
- c un tipo di dato
- d un commento
- 19 Lo stack pointer:
- a è decrementato durante l'esecuzione di una push
- b è decrementato durante l'esecuzione di una pop
- c è utilizzato per le istruzioni di salto (jmp)
- d è sempre gestito dal programmatore

## Domande per la prova orale

- 20 Che cos'è il linguaggio Assembly e qual è il suo legame con il linguaggio macchina?
- 21 Che cos'è il codice operativo e a che cosa servono gli operandi di un'istruzione?
- 22 Quali sono gli elementi che caratterizzano l'architettura del microprocessore DuplOne?
- 23 Che cosa significa «codice sorgente»?
- 24 Come è fatta una generica istruzione Assembly?
- 25 Che cosa significa «metodo di indirizzamento»?
- 26 Che differenza c'è tra un'istruzione jump e un'istruzione call?
- 27 Che cosa succede in fase di esecuzione dell'istruzione ret?

# Verifica delle ABILITÀ

## Unità 4

- 28 Scrivi un programma in DuplOne per sommare 10 numeri naturali pari.
- 29 Scrivi un programma in DuplOne per sommare 10 numeri naturali dispari.
- 30 Scrivi un programma in DuplOne tale che, dato un numero X in memoria, vengano letti N numeri in ingresso. Conta quanti di questi sono uguali a X.
- 31 Scrivi un programma in DuplOne capace di eseguire la seguente moltiplicazione:

$$Z = X \times Y$$

dove X, Y, Z sono il nome delle locazioni di memoria e K una variabile di appoggio.

*Suggerimento:* completa le istruzioni del programma seguente.

Algoritmo	Pseudocodice	Programma
Leggi X	Leggi X	molt: mov .....
Leggi Y	Leggi Y	mov .....
<b>U ← 0</b>	<b>U ← 0</b>	su: mov .....
<b>K ← Y</b>	<b>K ← Y</b>	mov .....
Ripeti: <b>U ← U + X</b> <b>K ← K - 1</b>	Ripeti: <b>U ← U + X</b> <b>K ← K - 1</b>	add ..... mov [Z], a mov a, [Y]
Finché K=0 Scrivi U	Se K=0 salta a Ripeti Scrivi U	dec ..... MOV [Y], a jnz .....
		halt

- 32 Con riferimento all'Assembly DuplOne, traduci in codice binario il seguente programma, specificando anche l'indirizzo di ogni istruzione e ipotizzando di allocare il programma a partire dall'indirizzo 0.

```
mov A,<LOC1>    loc1=03h
inc A            loc2=04h
add A,B          loc3=05h
mov B,<LOC2>
Decrementa: dec B
mov A,B
jnz decrementa
mov <LOC3>,A
```

- 33 Scrivi in linguaggio naturale il significato delle seguenti istruzioni scritte in DuplOne.

- a. mov A,24h
- b. add B,03h
- c. mov [24h],A
- d. inc B

- 34 Scrivi in linguaggio DuplOne le seguenti istruzioni espresse in linguaggio naturale.

- a. Trasferire nel registro A il contenuto della locazione di memoria di indirizzo 34h
- b. Trasferire nella locazione di memoria di indirizzo 26h, il valore 68h
- c. Sommare al contenuto del registro A il valore 64h e mettere il risultato in A
- d. Incrementare il valore del registro B



## The Assembly Software



### Abstract

Programming means writing a set of instructions (computer programs or software) that the CPU will use to implement an activity in a specific way. A software program is written with a programming language. A low level language is strictly dependent on a specific hardware and can be executed only on a specific computers family. A high level language is independent from a specific hardware and can be used on several computer types.

The «machine language» is the lower level language (written in binary). The Assembly language is a low level language strictly dependent on the computer architecture of the used machine.

Each Assembly instruction is a mnemonic representation of the machine operation code (what the machine must do) plus some operands if required. The instructions are classified according to their function.

### Questions

#### Answer all questions

- 1 What does CPU programming mean?
- 2 What's an high level language?
- 3 What is the format of a generic Assembly instruction?
- 4 How are the Assembly instructions classified?

#### Choose the correct answer



- 5 The operative code of an instruction:
  - contains the data
  - is the whole instruction
  - indicates what the CPU must do
  - is the address of a location memory

### Crosswords

#### Across

- 1 An Assembly instruction that jumps to some executable function and returns to the initial point
- 3 Type of instruction used to change the flow of control in a program
- 4 Mnemonic code for a sum instruction
- 6 The collection of programs used to control hardware
- 8 A low level language that uses mnemonics to reference machine code

#### Down

- 1 Used to measure two values against each other
- 5 Mnemonic code for memory operations
- 7 Translates Assembly code into binary code

#### 6 The call instruction:

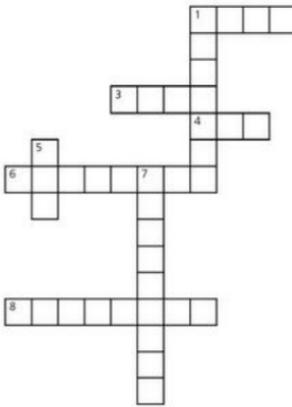
- executes a no return jump
- uses the stack
- executes a ret
- always tests the status register

#### 7 An algorithm:

- is a program written in machine language
- is a program written in natural language
- is a program written in high level language
- is a program written in low level language

#### 8 The phase to execute an instruction are:

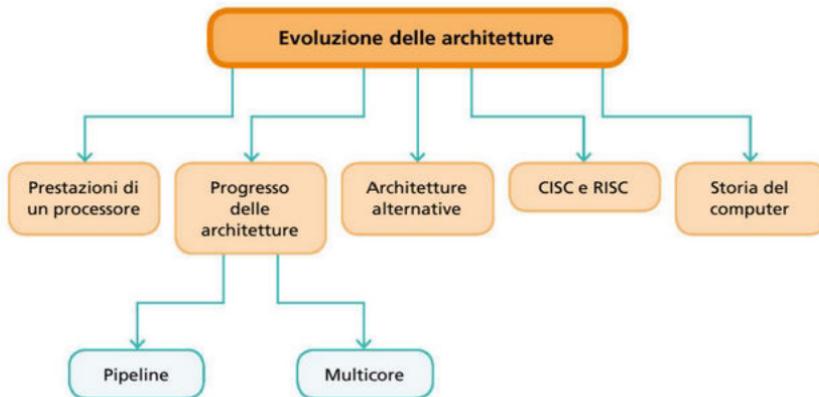
- fetch - execute - decode
- execute - fetch - decode
- decode - fetch - execute
- fetch - decode - execute





# Unità 5

## Evoluzione dei microprocessori



### Visione d'insieme

- I criteri per stabilire se un microprocessore è migliore di un altro.
- Le tecniche utilizzate per aumentare le prestazioni di un processore.
- Le diverse architetture con cui può essere progettato un sistema di elaborazione.

## 1 Introduzione

Nell'evoluzione sopravvivono e si procurano una discendenza solo quegli individui che meglio si adattano all'ambiente.

Anche i manufatti, seppure in forme diverse, sono soggetti a questa legge: devono migliorare per far fronte alle nuove esigenze degli utenti.

L'evoluzione delle architetture dei sistemi di elaborazione è stata determinata dalla richiesta di prestazioni migliori a un costo inferiore. Chi ci è riuscito, come per la famiglia dei processori x86, fa ancora parte della nostra vita. Degli altri ci siamo dimenticati.

In alcuni casi si sono superati i limiti fisici dei semiconduttori attuando dei «trucchi». Per esempio, per i processori che non possono lavorare oltre certe frequenze di clock, perché il calore prodotto danneggierebbe i circuiti, sono state progettate nuove architetture in grado di svolgere più programmi in parallelo (come in una catena di montaggio) facendo cooperare più processori tra loro.

## 2 Prestazioni di un sistema a microprocessore

Valutare le prestazioni (*performance*) di un sistema significa confrontare le prestazioni di quel sistema con quelle di un altro sistema che fa da riferimento, basandosi su parametri **misurabili**. Questa attività di misura è difficile: non basta considerare la velocità della CPU, ma occorre valutare anche il tipo di istruzioni, il sistema operativo, gli accessi alla memoria e ai dispositivi di I/O.

Di seguito vengono presentati alcuni criteri, tra i più comuni, usati per valutare le prestazioni di un sistema a microprocessore.

### Frequenza di clock

La **frequenza di clock** (o velocità di clock, o *clock rate*), influisce notevolmente sulla velocità di elaborazione.

La frequenza ( $f$ ) si misura in **hertz (Hz)**: 1 Hz corrisponde a un ciclo (di clock) al secondo (**Figura 1**). Il periodo ( $T$ ) è la durata di un ciclo:  $T = 1/f$  e si misura in secondi. Poiché i valori in gioco sono molto elevati, si prendono in considerazione i multipli dell'hertz, cioè il megahertz (MHz) e il gigahertz (GHz):

$$1 \text{ MHz} = 10^6 \text{ Hz}$$

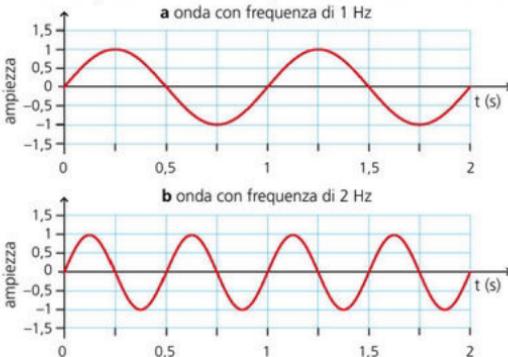
$$1 \text{ GHz} = 10^9 \text{ Hz}$$

Così una frequenza di clock di 1 MHz corrisponde a un periodo di un milionesimo di secondo e una frequenza di 1 GHz a un periodo di un miliardesimo di secondo.

### Pit Stop

- 1 Che cosa significa valutare le prestazioni di un sistema?
- 2 Che cos'è la frequenza di clock?

→ **Fig. 1** Onda (b) con frequenza di 2 Hz, doppia rispetto a quella di a, ma con la stessa ampiezza e la stessa fase.



Nei sistemi Windows, la frequenza di clock può essere ricavata dalle «informazioni sul sistema» (**Figura 2**).

① Specifiche dispositivo

Nome dispositivo	LAPTOP-A0IJFPU2
Processore	Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz
RAM installata	16,0 GB (15,8 GB utilizzabile)

◀ **Fig. 2** Informazioni di sistema Windows. La prima frequenza mostrata è la frequenza base del processore, la seconda quella calcolata da Windows.

In realtà la frequenza non è considerata un parametro particolarmente significativo, poiché indica solo la velocità con cui lavora il microprocessore, ma nulla dice in merito a quanti cicli occorrono per eseguire le istruzioni.

**MIPS (Million Instructions Per Second)**

È il numero (in milioni) di istruzioni al secondo che il microprocessore è in grado di eseguire; viene determinato nel seguente modo:

$$\text{MIPS} = \frac{\text{numero istruzioni}}{\text{(tempo di esecuzione in secondi} \times 10^6)}$$

Questo parametro valuta le prestazioni in funzione del tempo di esecuzione (le macchine più veloci hanno un MIPS più elevato) ma:

- non tiene conto delle caratteristiche delle istruzioni: se si confrontano architetture con set di istruzioni diverse, il valore sarà diverso;
- nello stesso calcolatore, i MIPS variano al variare del programma; infatti ogni programma può contenere istruzioni di diverso tipo che richiedono un tempo di esecuzione diverso.

**FLOPS (Floating Point Operations Per Second)**

È il numero di operazioni in virgola mobile eseguite in un secondo dalla CPU:

megaFLOPS MFLOPS  $10^6$   
 gigaFLOPS GFLOPS  $10^9$   
 teraFLOPS TFLOPS  $10^{12}$   
 petaFLOPS PFLOPS  $10^{15}$   
 exaFLOPS EFLOPS  $10^{18}$   
 zettaFLOPS ZFLOPS  $10^{21}$

Gli esempi che seguono mostrano come calcolare il tempo di esecuzione della CPU in funzione della frequenza di clock (**Esempio 1**) e del numero di istruzioni macchina di cui il programma è composto (**Esempio 2**).

### Esempio 1

#### Calcolo del tempo di esecuzione della CPU in funzione dei cicli di clock

##### Dati

- Frequenza di clock = 100 GHz =  $100 \times 10^9$  Hz;
- numero di cicli di clock per l'esecuzione di un programma =  $10^9$ .

##### Soluzione

Dalla frequenza ricaviamo il periodo: periodo di clock =  $\frac{1}{100 \times 10^9}$  s =  $10 \times 10^{-12}$  s = 10 ps

dove ps è il simbolo dei picosecondi ( $1 \text{ ps} = 10^{-12} \text{ s}$ ). Ora possiamo calcolare il tempo impiegato dalla CPU:

**tempo impiegato dalla CPU** = numero cicli di clock / frequenza di clock = numero cicli di clock × periodo di clock =  $10^9 \times 10 \times 10^{-12}$  s =  $\frac{1}{100}$  s

Per migliorare le prestazioni possiamo agire in due modi: diminuire il numero di cicli per eseguire un programma o aumentare la frequenza di clock. In questo caso, però, il numero di cicli necessario per eseguire le varie istruzioni potrebbe aumentare perché, per esempio, l'accesso alla memoria potrebbe causare dei ritardi.

### Pit Stop

- 3 Che cosa s'intende per MIPS?

## Esempio 2

**Calcolo del tempo di esecuzione della CPU in funzione dei cicli di clock e del numero di istruzioni del programma**

Possiamo ricavare misure utili definendo il **numero medio di CPI (Cicli di clock Per Istruzione)**:

**CPI** = (numero di cicli di clock impiegato dalla CPU per eseguire programma) / (numero istruzioni macchina del programma)

Il tempo della CPU (che era, vedi l'esempio precedente, numero cicli di clock × periodo di clock) diventa pertanto:

**tempo di CPU = numero istruzioni × CPI × periodo di clock**

### Dati

- Durata del periodo di clock = 10 ns;
- CPI = 2 (cicli di clock, in media, per eseguire un'istruzione);
- numero di istruzioni del programma = 10 000 istruzioni.

### Soluzione

$$\text{Tempo di CPU} = 10\,000 \times 2 \times 10 \times 10^{-9} \text{ s} = 200 \times 10^{-6} \text{ s} = 200 \mu\text{s}$$

### Pit Stop

4 Che cos'è un benchmark? A che cosa serve?

### Benchmark

In italiano *benchmark* significa «banco di prova». Questo termine indica un test usato per ottenere una valutazione attendibile e confrontabile. Vengono utilizzati alcuni programmi che fanno eseguire al sistema tutte le istruzioni del proprio set: meno tempo un sistema impiega a eseguire il benchmark, migliore è la prestazione.

Con il benchmark si misurano le prestazioni di un sistema e non del solo microprocessore, considerando i fattori riportati nella **Tabella 1**.

I test di benchmark assegnano dei punteggi alle diverse CPU per poterle confrontare tra loro.

↓ Tab. 1

Fattori che influiscono sulle prestazioni del sistema	
<b>Algoritmo utilizzato</b>	Un problema può essere formalizzato con algoritmi diversi. L'algoritmo influenza sul numero e sul tipo di istruzioni utilizzate.
<b>Linguaggio di programmazione e compilatore</b>	Entrambi influiscono sul numero di istruzioni a disposizione e sul numero di cicli macchina necessari per l'esecuzione di ogni istruzione.
<b>Architettura utilizzata</b>	L'architettura influenza sul numero di istruzioni a disposizione, sul numero di cicli macchina necessari per l'esecuzione di ogni istruzione e sulla velocità di esecuzione di ogni istruzione (cicli di clock).

### notabene

La massima temperatura di lavoro consentita per un componente elettronico è la temperatura massima alla quale quel componente può essere esposto senza subire danni o un degrado delle prestazioni.

## 3 Overclocking

Una tecnica, spesso applicata su PC domestici quando eseguono un videogioco, è l'**overclocking** che agisce sulla frequenza di clock per migliorare le prestazioni del **processore** o, più in generale, della **scheda grafica** e dell'intero sistema. Ogni modello di CPU prevede una frequenza ottimale di funzionamento, tuttavia è possibile indurre il processore a lavorare a una frequenza superiore a quella prevista, fermo restando il vincolo di non superare la massima temperatura di lavoro consentita.

Bisogna infatti tenere presente che l'overclock non è a costo zero, perché richiede un aumento del voltaggio del processore, che assorbe maggiore energia per dissipazione: il chip consuma di più, **genera più calore**, invecchia precocemente e ha una durata di vita ridotta.

Per evitare il surriscaldamento della CPU è bene dotarsi di adeguati dissipatori di calore (eventualmente a liquido), insieme a un'ottima ventilazione.

Anche se alcune CPU dispongono di tecnologie che aumentano la frequenza di clock in modo dinamico per gestire carichi di lavoro impegnativi, quasi tutti i processori di alto livello hanno la possibilità di intervenire sulla frequenza di clock accedendo ai parametri del BIOS presente sulla scheda madre.

In questo modo i programmi che richiedono più risorse (videogames, rendering 3D ecc.) potranno essere eseguiti in modo più veloce.

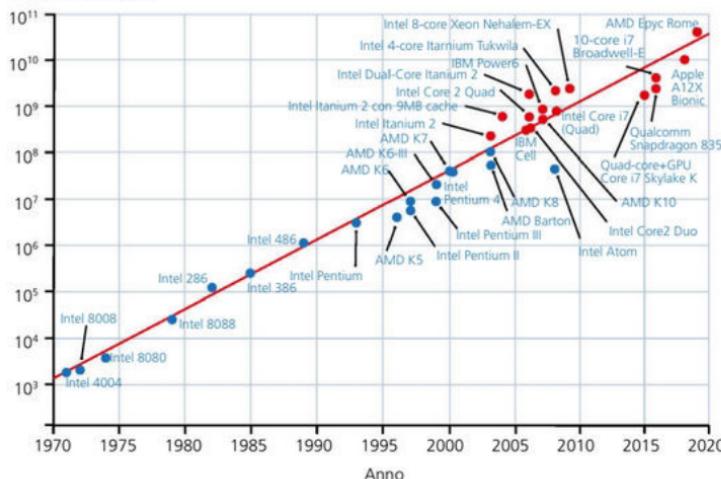
Conviene sempre tenere sotto controllo lo stato del sistema (temperatura e situazioni destabilizzanti) tramite appositi software di **stress-test**, come OCCT, che si trovano facilmente in rete.

## 4 Il progresso delle architetture dei sistemi di elaborazione

Nel 1965, Gordon Moore, cofondatore di Intel, enunciò quella che sarebbe diventata nota come Prima legge di Moore: «il numero di transistor contenuti in un chip raddoppia ogni 18 mesi». Questa congettura, fatta sulla base dei pochi dati disponibili allora, si è confermata valida fino a oggi, predicendo l'introduzione di calcolatori sempre più minuscoli, veloci ed economici.

La Figura 3 rappresenta graficamente la **prima legge di Moore**. Il grafico mostra una linea retta: sull'asse delle ascisse c'è l'anno di fabbricazione della CPU, sulle ordinate il numero di transistor presenti nei chip. La scala delle ordinate è logaritmica e il fatto che i dati tendano a disporsi lungo una linea retta significa che, per cinquant'anni, la crescita del numero dei transistor è stata esponenziale. Seguendo lo sviluppo tecnologico e sempre con l'obiettivo di migliorare le prestazioni del sistema, sono state introdotte altre tecniche che hanno impatto sull'architettura del sistema. Di seguito prenderemo in considerazione il pipeline e il multicore.

Numero di transistor



### Pit Stop

5 A che cosa serve l'overclocking?

6 A quali conseguenze negative porta l'overclock?

### Pit Stop

7 Enuncia la prima legge di Moore.

Fig. 3 La prima legge di Moore prevede che circa ogni anno, o poco più, il numero dei transistor contenuti nei chip raddoppi.

## 5 Pipeline

### Pit Stop

- 8 A che cosa serve la tecnica del pipeline?

La tecnica del pipeline consiste nel suddividere l'esecuzione di un'istruzione in fasi (sottoattività) ed eseguire in parallelo, cioè contemporaneamente, anziché in modo sequenziale, fasi di istruzioni diverse. Il tempo di esecuzione di ogni istruzione non cambia, ma **si riduce il tempo totale di esecuzione di un programma**. Ciò è possibile solo se si dispone di componenti autonome che possono eseguire una sottoattività indipendentemente dal resto del sistema.

Vediamo ora un esempio e un'analogia per capire meglio il significato di pipeline.

### Esempio 3

#### Parallelizzazione di sottoattività

Supponiamo di dover svolgere quattro attività, ciascuna scomponibile in 3 fasi:

A11, A12, A13  
A21, A22, A23  
A31, A32, A33  
A41, A42, A43

Per semplicità supponiamo che ogni fase richieda lo stesso tempo di svolgimento (ipotesi certamente poco realistica, ma accettabile in questo contesto).

- La **Figura 4a** mostra il tempo totale necessario per eseguire le quattro attività in **modo sequenziale**. Sull'asse orizzontale è rappresentato il tempo  $t$ . Il **tempo totale** necessario per eseguire le quattro attività è di 12 quanti di tempo.
- La **Fig. 4b** mostra invece il tempo necessario per svolgere le quattro attività parallelizzando, cioè svolgendo contemporaneamente, fasi di attività diverse. Il **tempo totale** necessario diminuisce notevolmente: in questo caso sono necessari 6 quanti di tempo contro i 12 dell'esecuzione in sequenza.

È molto importante sottolineare che si riduce il tempo totale di esecuzione, mentre il tempo necessario per l'esecuzione di un'unica attività rimane invariato (in questo caso 3 quanti di tempo).

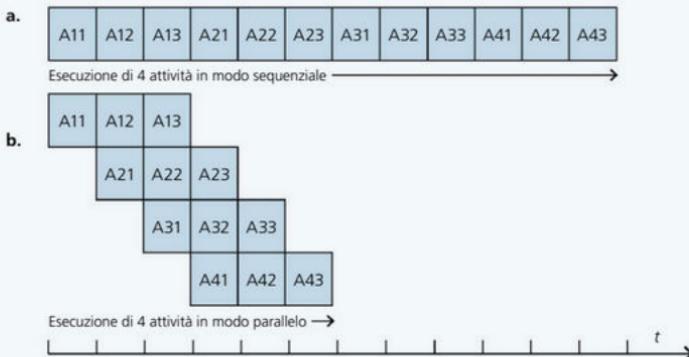


Fig. 4 Attività sequenziali e parallele.

## Comprendi con l'analogia

### Il pipeline è come una sessione d'esame

È molto frequente trovare esempi di parallelizzazione di attività nella vita reale, soprattutto laddove un obiettivo fondamentale è l'efficienza.

Immaginate una sessione di esami di Informatica. L'organizzazione ha a disposizione un solo computer. Nella sessione d'esame sono iscritti 4 candidati che vengono chiamati in ordine alfabetico: Alice, Bob, Carlo e Daria (Alice e Bob devono sostenere un esame sulla crittografia).

Per ogni candidato è possibile suddividere l'esame in 5 fasi:

1. Riconoscimento del candidato R



4. Prova orale O



2. Prova pratica con computer P



5. Esito finale (nella migliore delle ipotesi!) E



3. Correzione prova pratica C



Si noti come il tempo necessario per l'esame di un singolo candidato non varia, mentre diminuisce il tempo necessario per l'intera sessione d'esame.

Ovviamente, per poter operare in questo modo sono necessari 5 commissari, ciascuno addetto allo svolgimento di una fase e soprattutto autonomi e indipendenti.

## notabene

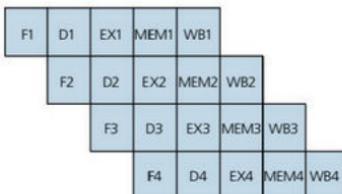
*Pipeline* in inglese significa «canale», «condotto». In informatica il termine pipeline è spesso usato per indicare qualcosa che entra in un sistema e, attraversando varie fasi, viene processato in modo che l'output di una fase sia l'input di quella successiva.

→ Fig. 5 Esecuzione di un programma di 4 istruzioni con pipeline a 5 stadi: dopo 4 cicli macchina, sono in esecuzione 4 diverse istruzioni.

Il compito del processore è quello di eseguire un programma costituito da più istruzioni, ciascuna suddivisa nelle tre fasi di fetch, decode, execute. Una generica istruzione può essere scomponibile in un numero maggiore di fasi. Il numero delle fasi dipende dall'architettura del sistema di elaborazione.

Per esempio, consideriamo una istruzione divisa in cinque fasi (o stadi) (Figura 5):

1. **F** (Fetch): lettura dalla memoria della prossima istruzione da eseguire;
2. **D** (Decode): decodifica dell'istruzione;
3. **EX** (Execute): esecuzione dell'istruzione;
4. **MEM** (Memory): attivazione della memoria (solo per determinate istruzioni, per esempio quelle di trasferimento). Nel caso in cui non ci sia alcun accesso alla memoria, l'istruzione viene fatta transitare fino allo stadio successivo;
5. **WB** (Write Back): scrittura del risultato in un registro.



Per poter eseguire contemporaneamente 5 fasi diverse, il processore deve essere suddiviso in 5 unità funzionali in grado di operare autonomamente e in modo il più indipendente possibile.

L'organizzazione è simile a quella di una catena di montaggio e consente l'esecuzione di più istruzioni (una fase per ciascuna) in un unico ciclo macchina. Il ciclo macchina è quello che nell'esempio abbiamo chiamato «quanto di tempo» ed è determinato dalla frequenza di clock.

In questo modo il processore viene utilizzato nel modo più efficiente possibile, in quanto a ogni ciclo macchina può iniziare una nuova istruzione e dopo 4 cicli processa un'istruzione per ciclo macchina. Questo accade in **condizioni ideali**, in cui il tempo necessario per l'esecuzione dell'intero programma è **ridotto di un fattore uguale al numero degli stadi**. Nella realtà il tempo risparmiato potrebbe essere inferiore. Questo dipende dalle istruzioni e dai dati, per cui alcune fasi potrebbero andare a vuoto. In ogni caso il tempo totale con pipeline rimane sempre inferiore al tempo necessario con un'esecuzione sequenziale.

## Esempio 4

### Efficienza del pipeline

Consideriamo un'architettura con pipeline a 5 stadi, cioè con 5 unità operative autonome, in grado quindi di eseguire contemporaneamente 5 fasi diverse di istruzioni diverse. Supponiamo che la CPU debba eseguire un programma costituito da 9 istruzioni. Per eseguire l'intero programma:

- senza pipeline, sarebbero necessari  $9 \times 5 = 45$  cicli macchina;
- con pipeline servono 13 cicli macchina.



Togliendo i primi 4 cicli necessari perché tutte le 5 unità entrino in funzione, il tempo totale per l'esecuzione del programma richiede 9 cicli macchina contro i 45 di un'architettura senza pipeline.

Questa è l'efficienza massima ed è possibile in una situazione ideale in cui, dopo i primi 4 cicli macchina, tutte le unità funzionano sempre e nessuna unità utilizza un tempo di ciclo macchina «a vuoto», senza poter lavorare.

Le prestazioni di un sistema in pipeline aumentano rispetto a un'architettura che lavora in sequenza (detta **scalare**), in funzione del numero di stadi previsti. Architetture dotate di più componenti funzionali dello stesso tipo possono realizzare più pipeline. Tali architetture sono chiamate **superscalari** e consentono un notevole aumento di prestazioni rispetto alle architetture scalari, in grado di eseguire un'unica istruzione alla volta.

Tornando alla precedente analogia sulla sessione d'esame, i tempi si accorcerrebbero ulteriormente, se la commissione d'esame avesse a disposizione un computer per ogni candidato e commissari sufficienti a seguire in parallelo le fasi d'esame di ogni studente.

### Problemi tipici del pipeline

La tecnica del pipeline presenta alcuni problemi tipici:

- alcune unità potrebbero avere la necessità di accedere alla stessa risorsa: per esempio, la memoria in fase di fetch e write back. In questo caso un'unità deve attendere, perdendo un ciclo macchina;
- si potrebbe verificare il caso in cui un'istruzione debba operare su un dato output di un'altra istruzione; il dato viene letto prima che venga modificato dall'istruzione;
- se durante l'esecuzione viene incontrata un'istruzione di salto o si verifica una interruzione, le operazioni già eseguite dalla pipeline potrebbero essere inutili. Per esempio, nel caso di salto (*IF... THEN... ELSE*) potrebbe essere stata fatta la fetch di un'istruzione che non è quella da utilizzare.

### Tecnica Branch Prediction

La **BPU** (*Branch Prediction Unit*) è una piccola memoria che mantiene una tabella con la storia degli ultimi indirizzi utilizzati nelle istruzioni di salto condizionato (per esempio, gli indirizzi delle istruzioni eseguite in seguito a una istruzione di *IF*). Quando viene eseguita un'istruzione di salto condizionato, l'istruzione successiva dipende dall'esito del test sulla condizione. La fetch anticipata potrebbe aver caricato una istruzione che non è quella che deve andare in esecuzione. Questa tecnica è basata sul calcolo delle probabilità: viene caricata l'istruzione che ha maggiore probabilità, in base alle esecuzioni precedenti, di essere eseguita. In caso di errore, cioè di errata previsione con conseguente caricamento di una errata istruzione, una soluzione potrebbe essere svuotare la pipeline. Nel caso peggiore, comunque, non si perde ulteriore tempo, e nel caso più favorevole se ne guadagna.

Negli ultimi anni i calcolatori che utilizzano la *branch prediction* sono stati oggetto di attacchi informatici (*Meltdown* e *Spectre*, Figura 6) che sfruttano alcune vulnerabilità dei moderni processori che lavorano su personal computer, dispositivi mobili e cloud, legate a fallo di progettazione.

### Pit Stop

**9** Che cosa s'intende per architettura superscalare?

**10** Che cos'è la BPU?



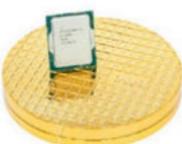
↑ Fig. 6 Il logo di Meltdown (uno scudo che si scioglie) e quello di Spectre (un fantasmino).

**Pit Stop**

- 11** Che cosa sono Meltdown e Spectre?

In particolare, Meltdown è una vulnerabilità hardware legata alla progettazione di molte moderne CPU Intel x86 e ARM (Spectre è una vulnerabilità simile che affligge microprocessori più recenti). Meltdown consente a un processo di «sciogliere» i confini della sicurezza imposti dall'hardware aggirando i normali controlli e accedendo allo spazio di memoria protetto, tramite un'analisi della cache dei dati durante l'elaborazione delle istruzioni nella pipeline e l'utilizzo della branch prediction. In questo modo un malintenzionato può ottenere dati riservati presenti nella memoria cache dei programmi in esecuzione: password, messaggi, foto, e-mail e documenti personali.

Nel 2018 degli attacchi che sfruttavano questa vulnerabilità hanno colpito molti server e dispositivi intelligenti basati su ARM, rallentandone il funzionamento.



↑ Fig. 7 Un processore Multi-Core Intel.

**Pit Stop**

- 12** Qual è il vantaggio dei processori Multi-Core?

**6 Multi-Core**

L'idea di disporre di **un unico chip contenente più processori (Multi-Core, Figura 7)** è nata dalla necessità di raggiungere prestazioni sempre più elevate, cosa che l'aumento della frequenza di clock o del numero di bit, da 32 a 64 a 128, non era più in grado di assicurare. Infatti l'aumento della frequenza del clock, e il conseguente aumento di velocità nelle operazioni, porta con sé un crescente ed eccessivo consumo di potenza elettrica, con relativo calore generato e difficilmente dissipabile, soprattutto nei sistemi mobili di dimensioni ridotte.

I primi processori composti da **doppio core (Dual-Core)**, realizzati da IBM, risalgono al 2003. La tecnologia si è rapidamente diffusa quando, nel 2005, Intel e AMD (tra i principali produttori di CPU) hanno presentato i loro primi modelli. I processori Multi-Core di ultima generazione possono contenere decine di core e lavorare a frequenze che superano i 5 GHz, offrendo prestazioni elevate sia per creazioni lavorative che per esperienze di gioco immersive.

Il numero di core installati è aumentato negli anni fino a superare i 70, un numero che, tuttavia, non è ancora sfruttato appieno dai sistemi software.

**7 Due diverse filosofie di progettazione delle architetture: CISC e RISC**

CISC e RISC sono modi di progettare l'architettura di un processore basati su filosofie diverse nel modo di intendere e gestire le istruzioni che possono essere eseguite da quel processore.

- Le architetture di tipo **CISC (Complex Instruction Set Computing)** sono caratterizzate da un insieme numeroso di istruzioni complesse che possono avvalersi di numerosi metodi di indirizzamento.

La complessità delle istruzioni si riflette sulla complessità dell'hardware del processore e, di conseguenza, sui tempi necessari per l'esecuzione delle istruzioni stesse.

Un'altra caratteristica importante di queste architetture è che la dimensione delle istruzioni è variabile in funzione del metodo di indirizzamento utilizzato. Questo ha un impatto sulla fase di fetch, che risulta essere più complessa, poiché non è noto a priori il numero di byte da prelevare.

La complessità penalizza le prestazioni, ma aumenta la scelta tra le istruzioni, diminuendo il divario esistente tra il linguaggio ad alto livello utilizzato dai programmati e il linguaggio macchina.

- Le architetture di tipo **RISC** (*Reduced Instruction Set Computing*) sono caratterizzate da poche istruzioni, semplici e di lunghezza fissa.

Operazioni complesse devono essere scomposte in operazioni elementari e gli operandi possono essere reperiti solo dai registri e non dalla memoria. Esistono delle operazioni specifiche per il trasferimento dati da/a memoria e registri (load e store). L'hardware è semplificato, così come sono ridotti i tempi di esecuzione delle istruzioni, ma le istruzioni sono meno potenti. La dimensione fissa certamente semplifica la fase di fetch e favorisce la realizzazione del pipeline.

Obiettivo di queste architetture è quello di avere a disposizione il minimo numero di istruzioni necessarie per risolvere qualsiasi problema, allo scopo di semplificare l'hardware e la gestione delle istruzioni.

### Pit Stop

- 13** Qual è la principale differenza tra l'architettura di tipo CISC e quella di tipo RISC?

## Comprendi con l'analogia

Le architetture CISC e RISC possono essere comprese con i mattoncini delle costruzioni

Utilizzare un'architettura RISC è un po' come avere a disposizione solo i mattoncini fondamentali delle costruzioni.

L'architettura CISC, invece, permette di avere anche pezzi di forme strane e diverse, le quali possono essere comunque ottenute combinando tra loro i pezzi base.

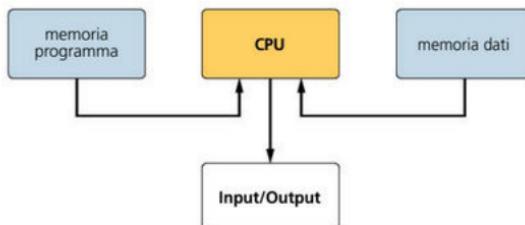


## 8 Architetture alternative nel gestire con la memoria dati e istruzioni

Sebbene negli anni siano state introdotte architetture diverse, come quelle dei *transputer* per il calcolo parallelo, le architetture che hanno avuto il maggior successo sono quelle di von Neumann, ancora oggi la più usata, e di Harvard.

La caratteristica dell'**architettura di von Neumann** è di avere un'unica memoria per dati e istruzioni e questo fatto potrebbe causare rallentamenti del sistema nelle operazioni sequenziali.

Nell'**architettura di Harvard** originale (Figura 8), che precede storicamente quella di von Neumann, i dati e le istruzioni risiedono in aree di memoria diverse. Questo fatto permette di leggere contemporaneamente un'istruzione e un dato parallelizzando le operazioni. La maggior velocità di esecuzione è a svantaggio di una elevata complessità.



◀ Fig. 8 L'originale architettura di Harvard. Successivamente l'architettura di Harvard è stata ripensata con un'area di memoria che contiene istruzioni e dati, e un'altra area di memoria che contiene solo dati.

L'architettura di Harvard è usata da alcuni microcontrollori ed è alla base dell'architettura dei **DSP (Digital Signal Processor)**, processori specializzati nell'esecuzione di calcoli utilizzati per il trattamento dei segnali digitali nei sistemi audio e video.

La **Tabella 2** confronta le caratteristiche principali delle due architetture.

von Neumann	Harvard
<ul style="list-style-type: none"> <li>• La memoria contiene dati e istruzioni accessibili nello stesso modo;</li> <li>• un solo bus rende semplice il controllo ed economica la realizzazione, ma diventa un collo di bottiglia per la gestione di dati, istruzioni e I/O;</li> <li>• la memoria può essere meglio sfruttata ripartendola tra dati e istruzioni;</li> <li>• questa architettura è utilizzata per tutti i computer di uso comune: desktop, laptop, dispositivi mobili.</li> </ul>	<ul style="list-style-type: none"> <li>• Due memorie con due <i>bus</i> consentono l'accesso contemporaneo a dati e istruzioni e rendono il sistema più veloce, ma anche più complesso e costoso;</li> <li>• le memorie possono essere prodotte con diverse tecnologie e avere celle di dimensioni diverse;</li> <li>• ci può essere uno spreco di memoria: la memoria dati può essere libera, ma non utilizzabile per le istruzioni e viceversa;</li> <li>• questa architettura è utilizzata principalmente per DSP e sistemi <i>embedded</i> (cioè, «incorporati», pensati per una specifica applicazione).</li> </ul>

→ Tab. 2

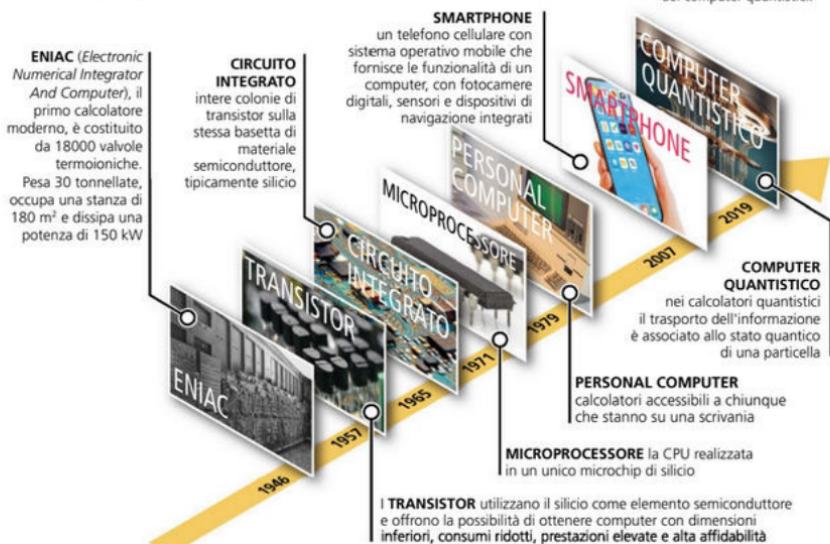
# Approfondimento - Tecnologia

## L'evoluzione dei calcolatori: dalle valvole ai computer quantistici

L'evoluzione dei computer segue un percorso, che è sintetizzato in **Figura 1**, che riporta le pietre miliari del cammino intrapreso dalla nascita del primo calcolatore fino ai nostri giorni.

ORIENTAMENTO  
**S T E M**

↓ Fig. 1 La timeline con le pietre miliari: dal primo computer all'ultima frontiera dei computer quantistici.



1946

**L'ENIAC** (Electronic Numerical Integrator And Computer), il primo calcolatore moderno (**Figura 2**), fu presentato nel 1946. L'elaboratore, progettato per scopi militari durante la seconda guerra mondiale, era costituito da **18000 valvole termoioniche** che erano soggette a frequenti rotture. Pesava 30 tonnellate, occupava una stanza di circa 180 m<sup>2</sup> e dissipava 150 kW di potenza.



1957

I successivi sviluppi tecnologici e le ricerche scientifiche portarono alla realizzazione di computer sempre più evoluti, ma la svolta avvenne nel 1957 con il lancio del primo computer commerciale che sostituiva le valvole con i **transistor**. I transistor utilizzano il silicio come elemento **semiconduttore** e offrono la possibilità di ottenere computer con dimensioni inferiori ai precedenti, consumi ridotti, prestazioni elevate e alta affidabilità.

**1965**

In realtà, non è stato il transistor in sé a determinare i rivoluzionari sviluppi dell'elettronica, dell'informatica e delle telecomunicazioni cui abbiamo assistito negli ultimi sessant'anni. Con i transistor si sono fatte le prime radioline e anche qualche elementare computer. A determinare lo sviluppo dell'elettronica moderna è stata, a metà degli anni Sessanta del secolo scorso, l'invenzione del **circuito integrato**, che ha permesso di contenere intere colonie di transistor sulla stessa basetta di materiale semiconduttore, tipicamente silicio. Ciò consente di transitare ai computer di **terza generazione**, caratterizzati da capacità di calcolo maggiori e dimensioni e costi di molto inferiori ai calcolatori della generazione precedente.

**1971**

Nel 1971 nasce il **microprocessore**: la CPU realizzata in un unico microchip di silicio che ha rivoluzionato per sempre la nostra vita!

In quegli anni Federico Faggin, un fisico italiano che aveva maturato esperienze precedenti sui circuiti integrati, progettò e realizzò il primo microprocessore Intel: il **4004**. Gli sviluppi seguenti allargarono la famiglia con i modelli 8008 e 8080, progenitori dell'8086 su cui sono basate ancora oggi le architetture x86 di Intel.

Con la **quarta generazione** inizia l'era dei *minicomputer* basati su microprocessore, come il DEC PDP-8 del 1974, che costituiscono una pietra miliare in campo informatico per le dimensioni e i costi ridotti.

**La parola a... Federico Faggin**

Federico Faggin nasce a Vicenza il 1º dicembre 1941. Diplomato nel 1960 presso l'Istituto Tecnico Industriale con specializzazione in radio-tecnica, si laurea in Fisica nel 1965 e si stabilisce negli Stati Uniti. Nel 1970 inizia a lavorare per Intel, progettando e realizzando il primo microprocessore: il 4004. Alla fine del 1974 lascia Intel e fonda la ZILOG, ideando e producendo Z80, processore a 8 bit, che ottiene molto successo. Il 19 ottobre 2010 gli è stata consegnata la Medaglia Nazionale per la Tecnologia e l'Innovazione dal Presidente degli Stati Uniti.

A proposito del 4004 seguiamo la nascita con le sue parole tratte dal libro «Silicio»:

«Finalmente giunse il gran giorno in cui ricevetti i primi wafer del 4004. Ecco arrivato il momento della verità! Ero molto teso. Per fortuna non c'era nessuno attorno a me che potesse vedermi in quello stato. Con mani tremanti misi il primo wafer sul prober, un apparecchio speciale per creare un contatto elettrico temporaneo sui chip che sono ancora integrati sul wafer. Abbassai le sonde sul primo chip, aspettando di vedere l'attività ormai familiare nel database, ma invece non accadde nulla. [...]»

Circa tre settimane dopo ricevetti una nuova serie di wafer del 4004. Questa volta nulla era stato trascurato, e me ne assicurai controllando subito un wafer sotto il microscopio prima di caricarlo sul prober. Respirai molto più liberamente dopo che i segnali familiari apparvero sull'oscilloscopio. Ora tutto stava procedendo meravigliosamente! Continuai a sondare fino a circa le 4 del mattino, riscontrando che tutto funzionava come previsto, finché, esausto, me ne tornai a casa. In quella notte di gennaio del 1971 nacque il primo microprocessore! Avevo da poco compiuto ventinove anni e mi resi conto che nove anni prima, a vent'anni, avevo finito di costruire un altro computer, realizzato con migliaia di transistor al germanio e altri componenti discreti. Quel computer aveva all'incirca le stesse caratteristiche di questo, con la differenza che il nuovo occupava una singola scheda invece di alcune centinaia, era dieci volte più veloce e consumava quasi mille volte meno energia. Per non parlare del costo irrisorio al confronto! Com'erano cambiate le cose in soli nove anni!»

Federico Faggin, «Silicio», Mondadori, 2020.

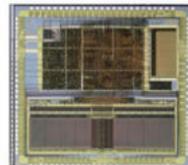


## 1979 e anni Ottanta

Gli anni Ottanta segnano il trionfo del personal computer che, ancora una volta, porta una rivoluzione nella fruizione dei computer: essi diventano **personal**i e accessibili a chiunque.

Negli anni seguenti si affermano anche i microcontrollori, costituiti da un circuito integrato che da solo può svolgere funzioni logiche e di calcolo e che ha a bordo memorie non-volatili per gestire i programmi applicativi. Un microcontrollore può costituire l'unità di controllo di un intero apparecchio (ad esempio di un elettrodomestico o di un utensile a controllo numerico) o di una sua parte (ad esempio l'ABS di un'automobile). Il caso estremo è rappresentato dalle smart card, nelle quali l'intero sistema è costituito da un singolo chip.

In **Figura 3** è riportato un microcontrollore della fine degli anni Novanta del secolo scorso che, meglio dei processori odierni, permette di identificare chiaramente regioni del chip: memoria e area di processo. Le regioni più ordinate (nella parte bassa della figura) sono aree di memoria, dove vengono immagazzinati i dati, mentre l'area in alto è la parte di logica di processamento dei dati.



↑ Fig. 3 Circuito integrato in tecnologia CMOS (microcontrollore). Fonte: STMicroelectronics.

## Anni 2000

I microprocessori sono entrati ovunque: dai personal computer fino agli smartphone (il primo iPhone è del 2007) e ai dispositivi intelligenti montati su auto e indossati insieme agli abiti.

L'incremento di **densità di integrazione** è stato reso possibile dalla riduzione esponenziale delle dimensioni dei componenti elementari. Le successive generazioni tecnologiche dei processi di fabbricazione dei circuiti integrati sono identificate dalla dimensione delle geometrie minime che possono realizzare.

Nel 1970 le dimensioni erano dell'ordine dei 10 micron o micrometri ( $1 \mu\text{m} = 1$  milionesimo di metro) e oggi sono inferiori ai 10 nanometri ( $1 \text{ nm} = 1$  miliardesimo di metro), si sono cioè ridotte di tre ordini di grandezza in cinquant'anni.

Anche sensori e attuatori hanno beneficiato del processo di miniaturizzazione che ha permesso di ridurre le loro dimensioni alla scala **micrometrica** ( $10^{-6} \text{ m}$ ). Nei **MEMS** (*Micro Electro-Mechanical Systems*, **Figura 4**) le parti meccaniche e i circuiti elettronici sono integrati sullo stesso chip di silicio.

Le maggiori ricadute in questo settore sono avvenute sui componenti utilizzati per le applicazioni **IoT** (*Internet of Things*) riducendo consumi e costi.

La riduzione delle dimensioni ha in realtà un limite fisico costituito dal fatto che un transistor non può funzionare se costituito da poche decine di atomi.

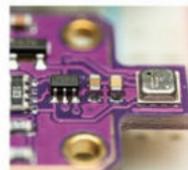
Per questo hanno fatto la loro comparsa nuove tecniche legate alla nanolettronica e ai computer quantistici.

Le **nanotecnologie** sono tecnologie applicate alla materia su scala nell'ordine del **nanometro** ( $10^{-9} \text{ m}$ ) e che coinvolgono molte discipline: fisica, chimica, informatica. In particolare la **nanolettronica** studia la possibilità di costruire circuiti integrati su scala molecolare. I campi di applicazione spaziano dai transistor realizzati con nanotubi di carbonio, ai sistemi per l'accumulo di energia, come le batterie in grafene, alle memorie, ai trasmettitori.

I **NEMS** (*Nano Electro-Mechanical Systems*) sono sistemi analoghi ai MEMS, ma su scala nanometrica.



Videointervista  
La nanolettronica



↑ Fig. 4 Il MEMS BME680 Bosch Sensortec, che si vede nell'estrema destra dell'immagine, è progettato per applicazioni mobili e dispositivi indossabili, combinando un sensore di gas per il rilevamento della pressione dell'aria, dell'umidità e della temperatura.

## 2019

I **computer quantistici** sfruttano i principi della meccanica quantistica, in particolare il dualismo onda-particella, per raggiungere una potenza di calcolo enormemente superiore a quella dei computer tradizionali.

# Approfondimento - Tecnologia

## ORIENTAMENTO S T E M

Fig. 1 Il computer quantistico di IBM Q System One, utilizzabile da remoto. Il processore superconduttore è un wafer simile a quello di un PC che è mantenuto a una temperatura vicina allo zero assoluto. Il computer completo ha le dimensioni di un'automobile ed è costituito, per la maggior parte, da sistemi di raffreddamento. I computer quantistici di IBM si programmano utilizzando Qiskit, un ambiente di sviluppo software, open source, che serve per eseguire calcoli quantistici utilizzando il linguaggio dei «circuiti quantistici», che consente di operare su dati quantistici come quelli contenuti nei qubit.



### I computer quantistici

I computer quantistici si basano su conoscenze teoriche particolarmente complesse e anti intutive.

Di seguito ne presentiamo le caratteristiche di base, senza avere la pretesa di entrare nei dettagli delle leggi fisiche della meccanica quantistica.

Mentre nei calcolatori tradizionali l'unità di informazione è il *bit* con i suoi stati binari, che corrispondono a due valori di tensione, nei calcolatori quantistici il trasporto dell'informazione è associata allo stato quantico di una particella o di un fotone. **L'unità di informazione è il *qubit* (*quantum bit*)** che, applicando il principio quantistico di **sovraposizione**, può assumere contemporaneamente sia il valore 0 che il valore 1. Con due qubit in sovrapposizione è possibile rappresentare quattro valori: 00, 01, 10, 11. Con tre qubit si rappresentano otto valori, e così via.

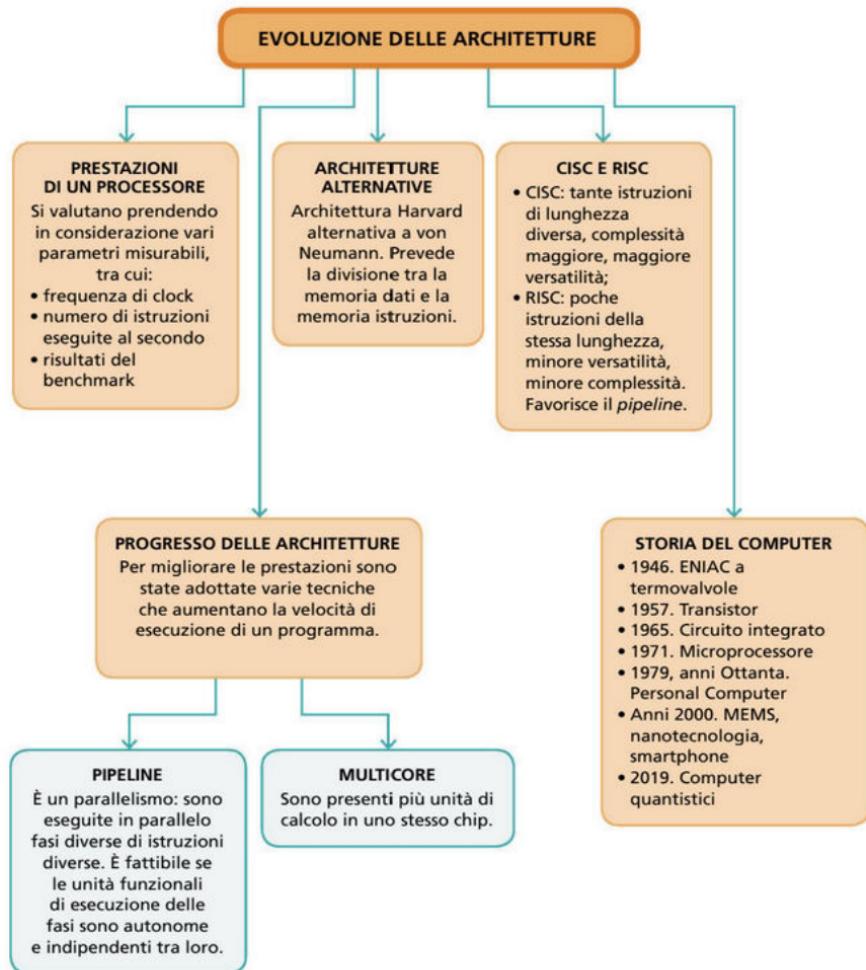
Lo stato esatto viene definito solo nel momento della misurazione, come insegnava il famoso esperimento mentale del gatto di Schrödinger chiuso in una scatola: prima di aprire la scatola il felino è contemporaneamente vivo e morto; l'apertura del contenitore fa «collassare» il gatto in uno dei due stati.

Quando due particelle sono prodotte insieme, danno origine a un sistema in cui si stabilisce una correlazione tra i loro stati quantici. La misurazione dello stato di una particella (ad esempio il suo spin) forza in modo opposto il valore dello stato dell'altra, indipendentemente dalla loro distanza. Questo fenomeno prende il nome di **entanglement**, intendendo che i destini delle due particelle restano tra loro indissolubilmente legati. Da ciò deriva che la misura dello stato di un qubit fornisce informazioni sullo stato degli altri qubit.

Da un punto di vista tecnologico molti di questi calcolatori sfruttano materiali **superconduttori** che, a temperature di qualche centesimo di grado sopra lo zero assoluto, hanno praticamente resistenza elettrica pari a zero.

Il calcolatore quantistico di IBM (la prima società a proporne uno sul mercato nel 2019, seguita da Google e Honeywell) utilizza la **giunzione Josephson** come elemento per contenere i qubit. La giunzione è costituita da uno strato isolante posto tra due superconduttori, come una fetta di prosciutto all'interno di un panino. La carica elettrica, trasportata da coppie di elettroni (**coppia di Cooper**), passa attraverso l'isolante grazie al fenomeno quantistico dell'**effetto tunnel**. Si pensi a una palla da tennis che, lanciata ripetutamente contro un muro, riesca occasionalmente a passarvi attraverso senza incontrare resistenza. Fasci di fotoni, sparati su questi qubit, ne controllano il comportamento e fanno in modo di trattenere e modificare le singole unità di informazione quantistica.

# SINTESI



# Verifica delle CONOSCENZE

Unità 5



## Vero o Falso?

- 1 Il clock è un'unità di misura.  
2 Il pipeline è una famiglia di processori.  
3 Nella fase di MEM viene scritto il risultato di un'istruzione.  
4 Il microprocessore migliore è sempre quello con velocità di esecuzione maggiore.  
5 Il set di istruzioni può incidere sulle prestazioni di un sistema.  
6 La frequenza è l'inverso del periodo ( $f = 1/T$ ).  
7 MIPS significa miliardi di istruzioni al secondo.  
8 Il benchmark è un test di valutazione che tiene conto del set di istruzioni.  
9 Il linguaggio di programmazione usato può influire sulla velocità di esecuzione di un programma.  
10 L'overclocking agisce sulla frequenza del clock.  
11 Il multicore utilizza più processori in parallelo.  
12 Il pipeline è una tecnica software.  
13 Tutte le architetture hanno pipeline a 5 stadi.  
14 Una architettura superscalare ha più pipeline.  
15 La Branch Prediction Unit è un'unità di calcolo dei sistemi multicore.  
16 In presenza di strutture di controllo il pipeline non è conveniente.  
17 La fase di write back scrive un dato in memoria.  
18 L'efficienza del pipeline è sempre pari al numero di istruzioni da eseguire.  
19 L'architettura CISC è più complessa della RISC.  
20 L'architettura RISC favorisce il pipeline.  
21 Nell'architettura RISC le istruzioni hanno tutte la stessa dimensione.  
22 CISC e RISC hanno lo stesso set di istruzioni.

V

F

V

F

V

F

V

F

V

F

V

F

V

F

V

F

V

F

V

F

V

F

V

F

V

F

V

F

V

F

- 23 L'architettura Harvard divide la memoria per i dati dalla memoria per le istruzioni.  V  F

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

- 24 La tecnica del pipeline:

- a non sempre è conveniente
- b necessita di tante unità funzionali autonome quanti sono gli stadi
- c è utilizzata solo in caso di branch prediction
- d è favorita dall'architettura CISC

- 25 Il benchmark:

- a è un test di valutazione per il confronto di diversi processori che tiene conto della frequenza del clock
- b misura il numero di istruzioni eseguite in un secondo
- c è un test di valutazione per il confronto di diversi processori che tiene conto di vari fattori
- d è la velocità di esecuzione di un programma

- 26 L'unità di misura della velocità di un processore si misura in:

- a bit al secondo
- b byte
- c hertz
- d secondi

- 27 L'architettura RISC:

- a è sempre più conveniente di un'architettura CISC
- b ha un set di istruzioni semplici
- c ha istruzioni di lunghezza variabile
- d è un'architettura non reale

- 28 La tecnica del pipeline:

- a ottimizza il tempo di esecuzione di un'istruzione
- b è utile se si opera su array
- c ottimizza il tempo di esecuzione di un programma
- d determina un eccessivo consumo di potenza elettrica

- 31 In che cosa consiste il branch prediction?

- 32 Quale è l'efficienza del pipeline?

- 33 In che cosa consiste il benchmark per la misura delle prestazioni di un sistema?

## Domande per la prova orale

- 29 Quali sono i vantaggi offerti dalla parallelizzazione delle attività e dal pipeline?  
30 Quali sono state le tappe fondamentali nell'evoluzione del computer?

### Rispondi alle seguenti domande

- 34** In riferimento alla parallelizzazione di sottoattività, scomponi il seguente problema della vita reale in sottoattività e proponi una gestione parallelizzabile, individuando le fasi, gli input e gli output di ogni fase, gli esecutori, eventuali problemi che si possono verificare. Discuti poi i vantaggi e gli svantaggi di tale organizzazione.

È una calda sera di agosto. Il piccolo comune di Torassi vuole organizzare una sagra dove verranno serviti pasta con sugo, polpette, salsiccia, vino, pane e acqua, se richiesta, e con un costo non incluso nello scontrino.

- Indicazioni preliminari
- Fasi
- Esecutori
- Problemi
- Vantaggi/Svantaggi

- 35** Proponi un ambito della vita reale in cui un processo può essere scomposto in sottoattività parallelizzabili e discutilo come nell'esercizio precedente.
- 36** Che cosa significa che una CPU lavora a 1 MHz?

- 37** Completa seguendo l'esempio.

1 kHz	equivale a	1000 Hz
1 MHz	equivale a	...
1 GHz	equivale a	...

- 38** Rileva il modello e la velocità di un computer che hai a disposizione.

- 39** Considera la seguente immagine:



Fai una ricerca sul web per capire il significato di tutti i codici.

- 40** Fai una ricerca sul web e riassumi, in forma tabellare, le principali CPU in commercio e le loro caratteristiche.

- 41** Fai una ricerca sul web e trova almeno due esempi di sistemi di elaborazione con architetture CISC e due esempi di sistemi di elaborazione con architetture RISC.



Esercizi in più



## CPU performance

### Abstract

There are many factors to consider when measuring the performance of a CPU. The most important ones are its clock frequency and its degree of parallelism, which is the ability to execute multiple instructions at the same time (pipeline).

Another way to improve the performance of a

computing system are multicore architectures, in which CPUs are mounted on a single chip. As time passed, computer architectures have evolved and CPUs have become both smaller and more powerful.

### Questions

#### Answer all questions

- 1 Describe the pipeline technique.
- 2 How are the microprocessor system performances measured?
- 3 Which conditions enable pipeline parallelism?
- 4 What does a multicore system entail?

#### Choose the correct answer



- 5 The RISC architecture:

- a has many instructions
- b is simple
- c is no longer used
- d has complex instructions

- 6 The pipeline:

- a is a system with multiple CPUs
- b executes instructions in parallel
- c is an evaluation test
- d is a system program

- 7 Quantum computers:

- a use qubits
- b were first developed in 2009
- c use transistors
- d do not use semiconductors

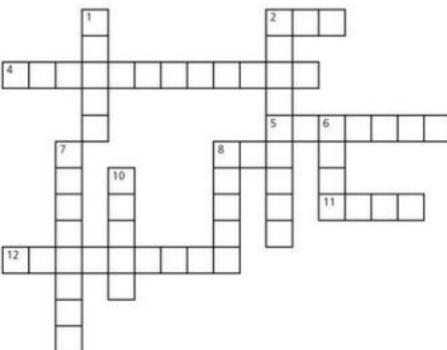
- 8 The pipeline:

- a reduces the execution time of an instruction
- b permits to execute one instruction at a time
- c slows down the system
- d reduces the execution time of a program

### Crosswords

#### Across

- 2 Memory that stores recent addresses used in jump instructions
- 4 Increases the clock speed
- 5 An alternative architecture to von Neuman
- 8 Memory access stage in a pipeline
- 11 An architecture with many complex instructions
- 12 System with multiple cores on a chip



#### Down

- 1 Measurement unit of the clock speed
- 2 An evaluation test used to check the performance of a system
- 6 An architecture with few simple instructions
- 7 Executes instructions in parallel
- 8 They made a famous prediction on the evolution of CPUs
- 10 A unit of quantum information

# Verifica delle COMPETENZE

## Tema 1

- 1** Leggi attentamente la seguente ricetta per preparare la marmellata di arance.

Pelate e tagliate a pezzetti 4 arance tenendo da parte un po' di scorza.

Pelate e tagliate a pezzetti 1 limone ed eliminate filamenti e semi. Mettete tutto in una ciotola unito a 400 g di zucchero. Cuocete mescolando fino a ebollizione. Mettete le scorze e dell'acqua in un pentolino e fate bollire per 5 min. Scolate le scorze e unitele alle arance. Fate cuocere per 15 min, frullate il tutto e mettete nei vasetti sterilizzati. Chiudete i vasetti e fateli raffreddare.



- a.** Individua e separa le istruzioni (verbi), i dati (ingredienti) e le risorse necessarie. Compila sul tuo quaderno una tabella strutturata in questo modo:

Istruzioni	Ingredienti	Risorse

- b.** Ordina le istruzioni numerandole: scrivi le singole istruzioni sul tuo quaderno, nella corretta sequenza ed evidenziando i verbi.

- 2** Descrivi i passi elementari che un sistema di elaborazione basato sul modello di von Neumann deve effettuare per calcolare il valore dell'espressione

$$(a + b) \cdot (c + d)$$

leggendo i valori delle variabili  $a$ ,  $b$ ,  $c$ ,  $d$  da un dispositivo di ingresso e scrivendo il risultato su un dispositivo di uscita.

- 3** Dopo aver individuato input e output, scrivi l'algoritmo per risolvere i seguenti problemi utilizzando una pseudo codifica:

- a.** Calcolare il valore dell'ipotenusa dati i due cateti di un triangolo rettangolo.
- b.** Date due numeri in input, fornire in output il massimo.
- c.** Dati in input prezzo e quantità di un prodotto, fornire in output il totale.
- d.** Dati in input due numeri e l'operazione (che può essere +, oppure -), fornire in output il risultato.
- e.** Dati in input tre valori, fornire in output il massimo.
- f.** Dati in input tre numeri, riscriverli in ordine crescente.
- g.** Calcolare il valore assoluto di un numero.
- h.** Calcolare il prodotto di due numeri con somme successive.

- 4** Classifica i seguenti sistemi in base alle caratteristiche:

- a.** Macchina distributrice di bevande

- b.** Catena montuosa

- c.** Roulette

### Rispondi alle seguenti domande.

- 5 Un computer è un sistema deterministico? Perché?
- 6 Che cosa si intende per architettura di un elaboratore?
- 7 Quanti byte possono essere indirizzati con un bus indirizzi a 32 linee?
- 8 Quanti bit di indirizzo servono per indirizzare 64 Kbyte di memoria?
- 9 Che cosa si intende per prestazioni di un sistema e come si misurano?
- 10 Quali sono le componenti interne di una CPU?
- 11 Quali sono i registri di uso generale di una generica CPU e quali sono le loro funzioni?
- 12 Quando può essere implementata la tecnica della pipeline?
- 13 Quali sono le principali differenze tra un'architettura CISC e un'architettura RISC?
- 14 Che cos'è un indirizzo di memoria assoluto?
- 15 In che cosa consiste il principio di località e quali vantaggi comporta?
- 16 Esponi la gerarchia di memoria con riferimento al trasferimento dati da un livello al livello adiacente.

- 17** Quali sono le principali caratteristiche dello stack?
- 18** Come sono collegate le periferiche al sistema?
- 19** In che cosa consiste la tecnica dell'interrupt per la gestione delle periferiche?
- 20** Proponi una situazione reale in cui è utilizzata una politica di gestione LIFO. Descrivi tale situazione identificando l'oggetto della gestione e motivando adeguatamente la risposta.

- 21** Consulta il web per determinare le caratteristiche delle ultime memorie di massa in commercio. Effettua un confronto di costi e prestazioni tra diverse case produttrici, compilando sul quaderno un prospetto che contenga:
- a. modello trovato
  - b. casa produttrice
  - c. caratteristiche
  - d. costo
  - e. prestazioni
  - f. confronto

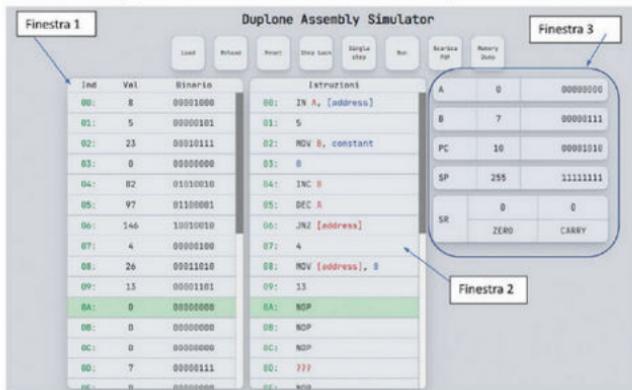
- 22** Traduci in binario le seguenti istruzioni assembly duplone:

- |              |                |
|--------------|----------------|
| a. dec B     | f. mov A , <5> |
| b. dec A     | g. jnz <05>    |
| c. mov A , B | h. jmp <05>    |
| d. mov B , A | i. cmp A , 5   |
| e. mov A , 5 |                |

- 23** Scrivi un programma duplone per codificare il seguente algoritmo

```
leggi N
leggi P
if (N != P) then
    N=N+1
```

- 25** Considera la seguente situazione. Osserva e descrivi il significato delle diverse finestre



```
else
    P=P+1
fine if
scrivi N
scrivi P
```

- 24** Considera la seguente situazione:

PC=00000001

Indirizzo	Contenuto RAM
00000000	10101010
00000001	00010110
00000010	00000111
00000011	00011110
00000100	00001100
00000101	00100001
00000110	00011001
00000111	00001010
00001000	00011110
00001001	00001010
00001010	10101010
00001011	10101010
00001100	00001000

La RAM contiene un programma e dei dati.

A partire dal contenuto del PC, la CPU esegue il programma che è costituito da 5 istruzioni.

- a. Traduci il programma in codice assembly DuplOne.
- b. Cosa contiene PC al termine del programma?
- c. Cosa contiene A al termine del programma?
- d. Cosa contiene B al termine del programma?

Memory Dump																			
00:	08	05	17	00	40:	00	00	00	00	80:	00	00	00	00	C0:	00	00	00	00
04:	52	61	92	04	44:	00	00	00	00	84:	00	00	00	00	C4:	00	00	00	00
08:	1A	00	00	00	48:	00	00	00	00	88:	00	00	00	00	C8:	00	00	00	00
0C:	88	07	00	00	4C:	00	00	00	00	8C:	00	00	00	00	CC:	00	00	00	00
10:	00	00	00	00	50:	00	00	00	00	90:	00	00	00	00	DC:	00	00	00	00
14:	00	00	00	00	54:	00	00	00	00	94:	00	00	00	00	D4:	00	00	00	00
18:	00	00	00	00	58:	00	00	00	00	98:	00	00	00	00	D8:	00	00	00	00
1C:	00	00	00	00	60:	00	00	00	00	9C:	00	00	00	00	DC:	00	00	00	00
20:	00	00	00	00	60:	00	00	00	00	A0:	00	00	00	00	EB:	00	00	00	00
24:	00	00	00	00	64:	00	00	00	00	A4:	00	00	00	00	E4:	00	00	00	00
28:	00	00	00	00	68:	00	00	00	00	A8:	00	00	00	00	E8:	00	00	00	00
2C:	00	00	00	00	6C:	00	00	00	00	AC:	00	00	00	00	EC:	00	00	00	00
30:	00	00	00	00	70:	00	00	00	00	80:	00	00	00	00	FB:	00	00	00	00
34:	00	00	00	00	74:	00	00	00	00	84:	00	00	00	00	F4:	00	00	00	00
38:	00	00	00	00	78:	00	00	00	00	88:	00	00	00	00	F8:	00	00	00	00
3C:	00	00	00	00	7C:	00	00	00	00	BC:	00	00	00	00	FC:	00	00	00	00

Rispondi alle domande:

- Che cosa rappresenta la finestra 1?
- Che significato hanno le colonne *Ind*, *Val*, *Binario*?
- Che cosa rappresenta la finestra 2?
- Che significato ha la finestra 3?
- Qual è la relazione tra i contenuti delle tre finestre?
- Che significato ha il contenuto di memoria evidenziato nella finestra Memory Dump?

# Orientamento al lavoro

ORIENTAMENTO  
STEM Tema 1

## Microprocessori

### Contesto

Impiego all'interno del team di ricerca e sviluppo dell'azienda leader mondiale nei settori networking e IT.

### Intervista a

*Domenico La Fauci*, cresciuto in Sicilia e laureatosi in ingegneria elettronica presso l'Università degli Studi di Messina. Adesso vive in Brianza con la moglie e i quattro figli. Dal 2001 fa parte del team di ricerca e sviluppo che si occupa della fotonica nell'azienda leader mondiale nei settori networking e IT.

**Quali sono i settori aziendali che si occupano di microprocessori e quale lavoro concretamente si basa sui microprocessori?**

La diffusione dei sistemi basati su microprocessori è ormai capillare. Si passa da quelli complessi presenti negli smartphone, nei PC, nei server, nelle telecomunicazioni, spesso multicore (cioè con più processori integrati in uno solo), a quelli più semplici, presenti per esempio in un apricancello radiocomandato, in un sistema di irrigazione o in un contatore di luce e gas.

Nei sistemi più complessi si utilizzano reti di microprocessori che possono avere una propria gerarchia di controllo, in maniera da suddividere

compiti e attività in base alle singole specificità. I microprocessori/microcontrollori, infatti, oltre a essere dotati di CPU e memorie, sono ormai ricchi di periferiche, per cui si assiste al proliferare di un'ampia varietà di microprocessori in cui si mantiene invariato il core (ARM, X86, MIPS, AVR, ...) e che si arricchiscono di periferiche di comunicazione (USB, I2C, SPI, UART, CAN) o di conversione (ADC, DAC, I2S, HDMI) necessarie per dialogare con il resto del sistema.

### Chi sono i clienti e i fornitori?

Chiunque voglia costruire un sistema embedded parte dalla scelta del microprocessore da utilizzare, dalle interfacce e dalle periferiche che compongono il sistema. Esistono molti fornitori di microcontrollori con microprocessori nella fascia 8 bit e 32 bit: *Microchip, ST, Renesas, Cypress, Analog Devices* sono tra i più noti.

Normalmente i clienti arrivano dal progettista con un'idea delle funzionalità che dovrà avere il sistema che desiderano. Sta al progettista trovare il giusto compromesso tra costi e potenzialità offerti dai microprocessori presenti sul mercato, in maniera da disegnare un prodotto che una volta progettato/testato/ingegnerizzato possa rispondere alle aspettative per cui è stato pensato.

### Qual è il mercato oggi e qual è la previsione per i prossimi anni?

Quando mi trovai tra le mani il mio primo microprocessore, mi parve qualcosa di incredibilmente potente e promettente. Si trattava di uno Z80, un oggetto che, se messo a confronto con i microprocessori presenti oggi sul mercato e accessibili con un semplice clic del mouse, fa quasi tenerezza!

La miniaturizzazione ha raggiunto oggi traguardi impensabili solo dieci anni fa ed è un settore in continua evoluzione. Un numero sempre maggiore di funzionalità viene integrato in spazi sempre più contenuti, rendendo i microcontrollori sempre meno costosi e sempre più multifunzionali e semplici da programmare.

Anche il software ha fatto enormi progressi: si è passati dal linguaggio assembly ai linguaggi ad alto livello come C, C++, Python, Java e addirittura a linguaggi visuali e a blocchi, in grado di esprimere concetti e funzionalità che si approssimano alle dinamiche del pensiero umano e sono in grado di trasferirle nel linguaggio macchina proprio del microprocessore.

Spesso si sente parlare di *firmware* e di *software applicativo* per indicare rispettivamente il software più a basso livello (cioè a livello del microprocessore) e quello più ad alto livello o *applicativo*, che spesso poggia su un sistema operativo (Android, Linux, Windows, macOS ecc.) e che definisce le funzionalità del sistema e l'interfaccia verso l'utente.

Il firmware mette il microprocessore in condizione di gestire tutte le periferiche o i blocchi funzionali a esso connessi. Sono spesso funzioni elementari che verranno poi armonizzate e completate dal sistema operativo o dall'applicazione.

Il dilagare del concetto di *IoT (Internet of Things)* ha dato un forte impulso allo sviluppo dei dispositivi basati su microcontrollori, spalancando le porte a nuovi campi applicativi, ricerche e sperimentazioni. Le performance di questi dispositivi sono oggi sempre più sofisticate.

### Qual è la figura professionale che è richiesta in questo ambito?

Lo sviluppo di sistemi basati su microprocessori avviene su più livelli: ci sono una fase progettuale, una fase di prototipazione, una di testing e messa a punto, una di produzione. La progettazione è mirata all'integrazione funzionale in sistemi complessi. Per intenderci, quando progetti un dispositivo intelligente e iperconnesso, devi aver chiaro lo scopo, cioè devi avere un'idea abbastanza precisa di che cosa dovrà essere in grado di fare. La prototipazione è una fase molto creativa: si tratta di risolvere decine di problemi e criticità e di rendere operativo ciò che si è pensato durante la progettazione. Il testing richiede precisione ed è fondamentale per garantire un buon prodotto affidabile.

Per poter operare efficacemente in questi ambiti sono richiesti tecnici e ingegneri altamente qualificati. Figure professionali dotate di ampie e solide competenze di base e allo stesso tempo capaci di innovazione. Gente, in buona sostanza, che sappia non solo applicare delle formule, ma anche progettare qualcosa di nuovo, gente *capace di una visione*.

### Che tipo di formazione deve avere?

La formazione offerta dagli Istituti Tecnici e dai corsi di laurea in Ingegneria Elettronica degli atenei italiani è mediamente di ottimo livello. Per acquisire le competenze specifiche richieste sul campo è tuttavia assolutamente necessario quello che si definisce «*training on the job*», ovvero la formazione in affiancamento. Questo per due ragioni

principal: la continua evoluzione tecnologica e l'evoluzione delle richieste del mercato.

A questo scopo le aziende creano gruppi di lavoro eterogenei, formati da collaboratori molto giovani e collaboratori di esperienza, in modo tale che i primi possano imparare dai secondi.

Va da sé che in questo mestiere sono indispensabili una buona capacità di relazionarsi, quasi sempre in lingua inglese, e grande disponibilità a lavorare in gruppo. Fondamentale è anche viaggiare per visitare fiere e distretti produttivi.

### **Ci racconta un po' la sua esperienza, la sua storia, il rapporto con i colleghi?**

Quello del *problem solver*, del *creatore di innovazione*, del *costruttore di cose nuove* che facilitano la vita quotidiana, non è solo un mestiere – è una *formam*.

Esiste un popolo di «smanettatori», meglio noti come *makers*, i quali, acquisite le nozioni fondamentali di elettronica, nella penombra umida delle proprie cantine, si dedicano alla progettazione e realizzazione di oggetti controllati elettronicamente, il cui aspetto e le cui funzionalità possono essere banali o assolutamente geniali.

Esistono dei luoghi di ritrovo dove questi personaggi si incontrano, si confrontano, condividono progetti e conoscenze. Li chiamano *fab lab* e sono delle vere fucine di idee e brevetti.

Esistono, infine, degli eventi cui i makers partecipano, una rete internazionale di *Maker Faire* distribuita nei cinque continenti. Sono frequentati non solo da moderni inventori, ma anche da gente comune, che desidera farsi sorprendere dalle diavolerie (a volte utili, a volte proprio no!) partorate dalle fertili menti dei makers.

Questi eventi sono molto interessanti anche per chi, nel settore dei sistemi di controllo basati su microprocessori, ci lavora. Il clima che vi si respira è quello degli eterni adolescenti, anche se c'è un

mucchio di gente che ha la barba bianca e proviene da terre lontanissime. Qui gli stimoli e gli spunti di ispirazione non mancano mai.

### **La cosa più bella e quella più brutta della sua esperienza lavorativa?**

Sviluppare sistemi basati su microcontrollori per applicazioni industriali o *smart city* significa trovarsi di fronte potenziali clienti che hanno un problema di tipo operativo nel sistema con il quale lavorano e ti sfidano a risolverlo.

Esempi classici sono la linea di produzione in cui un certo step deve essere sensorizzato, o uno spazio pubblico dove si vuole monitorare il flusso di automezzi o persone.

Una volta raccolta la sfida, il bello è vincerla. Trovare, cioè, la soluzione più efficace, flessibile, economica e, probabilmente, anche scalabile! Realizzare un dispositivo che risponda alle esigenze del cliente e migliori davvero il suo lavoro.

Per ottenere questo risultato a volte ci passi le notate! La soluzione non è sempre a portata di mano e si possono imboccare strade sbagliate; quindi si deve ricominciare daccapo. Bisogna essere ostinati, in questo mestiere, e avere fiducia nel fatto che «se lo puoi pensare (quasi sempre) lo puoi anche fare»!

### **È contento del suo lavoro? Lo sceglierrebbe di nuovo?**

Poche cose sono più gratificanti di un progetto che funziona e risolve un problema operativo. Il nostro lavoro è terribilmente stimolante. Offre la possibilità di mettersi alla prova, di sperimentare la frustrazione dell'insuccesso e di imparare a gestirla. Educa a osservare la realtà da punti di vista diversi e a cercare strade nuove per affrontarla, vie poco battute o da tracciare di sana pianta.

Infine è un lavoro che ha quasi sempre l'obiettivo di costruire qualcosa di utile a migliorare la vita di qualcun altro, e nel farlo ci si diverte un mondo!

# Sfida finale

ORIENTAMENTO  
S T E M

Tema 1

## Un computer per il gaming

### Scenario

Hai avuto dei bellissimi voti quest'anno e i tuoi genitori vogliono regalarti un nuovo computer adatto per il gaming, di cui sei un grande appassionato. Ti chiedono di fornire loro i nomi dei componenti che sarai tu a montare nel PC desktop, badando bene al rapporto prezzo/prestazioni e ai consumi energetici.

Tu sei molto felice di soddisfare le loro attese e decidi di preparare il documento con i modelli dei componenti che soddisfano le tue esigenze, tenendo presente che un computer per il gaming deve avere caratteristiche specifiche.

In particolare:

- un processore potente, *multicore*, per supportare rendering 3D e *frame rate* elevati;
- memorie RAM di grandi dimensioni e molto veloci;
- dischi veloci o, meglio ancora, SSD;
- schede video con GPU specializzate, con grandi quantità di memoria video e molto veloci;
- sistemi di raffreddamento potenti ad aria e a liquido;
- accessori specifici per il tipo di gioco più utilizzato: microfoni, cuffie, occhiali, oltre a uno o più monitor.



### Obiettivi

Preparare una tabella con la configurazione dei componenti del PC, specificando:

- |                     |                          |
|---------------------|--------------------------|
| • <b>Modello</b>    | • <b>Caratteristiche</b> |
| • <b>Produttore</b> | • <b>Costo</b>           |

Motivare le scelte con una breve nota di spiegazione.

### Consegna

**1 Consultazione** Consultare le pubblicazioni in Internet, compreso il sito ufficiale del produttore, oppure rivolgersi a un punto vendita locale, raccogliendo tutte le informazioni utili per assemblare il sistema. Integrare le informazioni con notizie informali ricevute da amici e compagni che hanno esperienza di giochi e sono informatissimi sulle ultime versioni e sulla loro «voracità» in termini di memoria e CPU.

### 2 Compilazione della tabella

	Modello	Produttore	Caratteristiche	Costo
CPU				
RAM				
Memoria di massa (disco o SSD)				
Scheda video				
Scheda audio				
Sistema operativo				
Raffreddamento				
Accessori (monitor, speaker, backpack ecc.)				

Note

---



---