09-09-2016

Sistemi di numerazione

La trasformazione di un **numero binario in un numero decimale** avviene secondo la seguente regola: **si moltiplica ciascuna cifra binaria a partire da destra per la corrispondente potenza di 2 e si sommano i prodotti ottenuti**
Per esempio ho il numero binario 11010, si ha: **$0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 = 26$**

Quozienti	Resti
70(:2)	0
35	1
17	1
8	0
4	0
2	0
1	1

La trasformazione inversa, da **numero decimale a di un numero binario**, viene effettuata secondo la seguente regola:
Si divide il numero dato per 2 e si scrive il resto (che può essere 0 e 1); il quoziente ottenuto viene a sua volta diviso per 2 ottenendo un nuovo resto; si prosegue fino a quando si ottiene come quoziente il numero 0.

La sequenza dei *resti*, letta dall'ultimo al primo, fornisce il n. binario corrispondente al n. decimale dato.
Es. la trasformazione del n. 70: **$70_{10} = 1000110$**

Quozienti	Resti
1602(:8)	2
200((,25):8)	0
25(:8)	1
3	3

Trasformazione del numero ottale 325 in numero decimale: $5 \times 8^0 + 2 \times 8^1 + 3 \times 8^2 = 5 + 16 + 192 = 213$
Trasformazione del numero decimale 1602 in numero ottale: $1602_{10} = 3102_8$
Trasformazione di un numero da esadecimale 3AF2 in numero decimale:$2 \times 16^0 + 15 \times 16^1 + 10 \times 16^2 + 3 \times 16^3 = 15090$

Binario	Esadecimale
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Regole di conversione binario/esadecimale - Nelle operazioni di conversione dei numeri dal sistema in base 2 al sistema in base 16, è opportuno ricordare la tabella:

Modello di trasformazione da binario a esadecimale
Si raggruppano le cifre del n. binario a gruppi di quattro a partire da destra, e si trasformano le cifre di ciascun gruppo nel corrispondente n. esadecimale, secondo la tabella di conversione.
Esempio il n. binario:1011110111si può scrivere come:

10	1111	0111
2	F	7

Quindi: **$1011110111_2 = 2F7_{16}$**

La cifra binaria 1100111110010111 corrisponde alla cifra esadecimale: CF97

Modello di trasformazione da esadecimale a binario
Si fa corrispondere a ciascuna delle cifre esadecimale che compongono il n. un gruppo di *quattro bit* raggruppano le cifre del n. binario a gruppi di quattro, secondo la tabella di conversione. Esempio il numero esadecimale: C3B si può scrivere:

C	3	B
1100	0011	1011

Quindi: **$C3B_{16} = 110000111011_2$**

Regole di conversione binario/ottale

Modello di trasformazione da binario a ottale
Si raggruppano le cifre del numero binario a gruppi di *tre* a partire da destra, e si trasformano le cifre di ciascun gruppo nel corrispondente numero ottale. Esempio il numero binario: 1011110111 si può scrivere come: **$1011110111_2=1637_8$**

1	011	110	111
1	3	6	7

Modello di trasformazione da ottale a binario
Si fa corrispondere a ciascuna delle cifre ottali che compongono il numero un gruppo di *tre bit.*
Esempio, il numero ottale: 625 si può scrivere:

6	2	5
110	010	101

Quindi: **$625_8 = 110010101_2$**

Un numero è **rappresentato dalle cifre 111 nel sistema ottale. In un altro sistema di numerazione ha la rappresentazione 201. La nuova base di numerazione è:** x=6; Infatti: $1 \times 8^0 + 1 \times 8^1 + 1 \times 8^2 = 73$.
Mentre 201 è: $1*x^0 + 0*x^1 + 2x^2 = 73 \rightarrow 2x^2 + 1 = 73 \rightarrow x^2 = 72/2 \rightarrow x = 6$.

0.456 E + 13 **La rappresentazione in atto, si chiama notazione esponenziale normalizzata, mette in evidenza la mantissa (la parte prima di e) e l'esponente (la parte dopo di E) che assume il nome di caratteristica. La caratteristica di un logaritmo** è la **parte intera;** La **parte decimale è la mantissa.**

L'informatica è la scienza della rappresentazione e dell'elaborazione dell'informazione.

L'informatica è lo studio sistematico degli algoritmi che descrivono e trasformano l'informazione: la loro teoria, analisi, progetto, efficienza, realizzazione e applicazione.

Per algoritmo (processo risolutivo) si intende la descrizione in modo informale di una sequenza finita di istruzioni (definite con precisione), che conducono alla realizzazione di un dato compito e per raggiungere un risultato definito in precedenza.

Esempi sono l'esecuzione di algoritmi per il calcolo del massimo comune divisore di diversi numeri naturali, le indicazioni per la consultazione on line di un orario di voli charter, ecc.
È essenziale che un algoritmo sia comprensibile al suo esecutore (macchina, ente o persona). Gli algoritmi sono descritti tramite **programmi**, cioè sequenze finite di istruzioni scritte in un opportuno linguaggio, comprensibile all'elaboratore. Il ruolo dell'esperto informatico è quello di realizzare procedimenti, cioè analizzare la sequenza di passi che conducono alla risoluzione di un problema e codificarli in programmi (cioè renderli comprensibili all'elaboratore).

Introduciamo ora due proprietà essenziali degli algoritmi: la *correttezza* e *l'efficienza*.
Un algoritmo si dice *corretto* qualora consente di pervenire alla soluzione del compito cui si è preposto, senza che presenti difetti o errori nella sequenza di passi.
L'altra proprietà auspicabile degli algoritmi è che siano *efficienti*, cioè di pervenire alla soluzione del problema nel modo più veloce possibile utilizzando il minimo di risorse hardware, compatibilmente con la sua correttezza.

Cos'è un computer o un elaboratore elettronico (digitale)?

Introduciamo la seguente definizione: diremo che un calcolatore elettronico è una macchina complessa utilizzata per la rappresentazione, l'elaborazione e la memorizzazione delle informazioni. Esso lavora partendo da informazioni in ingresso (l'input del processo di elaborazione), la elabora in base a una serie di istruzioni assegnate (programma), e restituisce informazione in uscita (l'output del processo). La quasi totalità dei computer oggi utilizzati è **digitale**, lavora cioè con informazioni "convertite in numeri" o meglio "rappresentate in forma numerica" (dall'inglese Digit = Cifra).

Un insieme limitato di istruzioni viene detto linguaggio macchina, e rappresenta il modo per comunicare con il computer.
Soluzione: livelli di astrazione, ognuno costruito sulla base di quello sottostante.
Procedura: insieme di istruzioni più idonee al programmatore per l'utilizzo rispetto al linguaggio macchina.
Compilazione: traduzione di tutte le istruzioni del linguaggio superiore in istruzioni del linguaggio inferiore. Si ottiene un programma eseguibile.
Interpretazione: si tratta di un programma (interprete) che effettua la traduzione delle singole istruzioni contestualmente alla sua esecuzione.
Osservazione: il linguaggio interpretato può presentare errori logici dovuti a forme sintattiche o semantiche errate.

Ambiente di programmazione

I linguaggi di programmazione sono molto diversi fra loro, ciascuno di essi è caratterizzato da un *proprio ambiente di programmazione*, cioè un insieme di strumenti che facilitano la scrittura dei programmi e la verifica della loro correttezza. Gli ambienti di programmazione, differenti fra loro, hanno molti aspetti in comune, quali un *editor*, un *compilatore*, un *linker* e un *controllore* dell'esecuzione.

- La finalità degli **editor** di un ambiente di programmazione consiste nel costruire *programmi sorgente*, cioè programmi scritti in un linguaggio di programmazione di alto livello.
- Il **compilatore** opera la traduzione di un *programma sorgente* (istruzione per istruzione) in un *programma oggetto* (**object program -** programma trasformato in un linguaggio pseudo macchina).

Se nel corso della traduzione, il compilatore trova degli **errori formali** (lessicali o sintattici), cioè scopre che delle istruzioni dell'algoritmo non sono corrette, allora in questo caso, provoca l'interruzione dell'esecuzione e il programma oggetto non viene generato. Tuttavia il programmatore viene però informato sulla natura degli errori trovati dal compilatore, e assistito nella correzione del programma sorgente.
Così come con la compilazione non è possibile rilevare **errori logici** che potrebbero verificarsi durante l'esecuzione del programma (**runtime errors**) sulla base di particolari valori assunti dalle variabili durante l'elaborazione: basti pensare ad esempio alla divisione per un numero che assume durante l'esecuzione il valore **0**.
Gli errori (formali) possono riguardare l'uso dei termini non appartenenti al linguaggio (*errori di tipo lessicale*), oppure la costruzione di frasi non corrette dal punto di vista delle regole grammaticali del linguaggio (*errori di tipo sintattico*).

- L'**interprete** esegue direttamente il codice sorgente (istruzione per istruzione) contestualmente alla sua traduzione (senza tradurlo nel linguaggio macchina). L'interpretazione, spesso viene utilizzata in alcuni linguaggi per la gestione di basi di dati.

Eventuali errori formali vengono rilevati e segnalati solo quando l'istruzione errata viene tradotta e causano l'interruzione dell'esecuzione.

- La compilazione deve essere seguita da un'ulteriore operazione detta **linker** (*collegatore*). Tale operazione consiste nell'aggiungere al programma compilato (moduli programma oggetto) i moduli del compilatore coordinati fra loro, i quali realizzano:

a) Funzioni richieste dai vari comandi;
b) Risolvono uno specifico problema applicativo (riferimenti a celle di memoria o a variabili).

Alla fine del processo si ottiene il programma eseguibile.

Differenze di efficienza tra la compilazione e l'interpretazione.
La compilazione consente di avere file detti **oggetto**, che non sono ancora eseguibili, mentre con l'interpretazione otteniamo direttamente il file eseguibile. L'interpretazione non richiede tempi di compilazione, ma, quando il programma viene eseguito, ha tempi di trasformazione che lo rendono molto più lento di un programma eseguibile. Per questi motivi l'interpretazione risulta utile nella fase di progettazione, mentre la compilazione è preferibile per un programma applicativo.

- Un **controllore** dell'esecuzione (**debugger**), cioè uno speciale programma di utilità che durante l'esecuzione del programma, serve ad individuare ed eliminare eventuali errori logici.

Esso può per esempio consentire di:

a) Eseguire il programma (un'istruzione per volta), in modo tale da verificarne la corretta evoluzione;
b) Controllare i valori assunti dalle variabili in certi punti dell'esecuzione.

Approfondimenti
Differenza tra linguaggi compilati e linguaggi interpretati.

Linguaggi compilati
Per tradurre le istruzioni si usa uno specifico programma che si chiama *compilatore*. Il compilatore si aspetta di ricevere un file sorgente scritto in caratteri ASCII e fornisce un file, detto *file oggetto*, che contiene la traduzione delle istruzioni in linguaggio "*pseudo macchina*". Il compilatore è specifico del linguaggio e del computer; questo significa che esiste un compilatore per ogni linguaggio ma, anche per ogni specifica macchina (es. IBM, DIGITAL, Macintosh, ecc…).
Esiste anche il concetto di *cross-compilatore* con il quale si intende un compilatore che *gira* su di un certo computer ma che, mediante opportuni comandi, è anche in grado di produrre file oggetto validi per altri tipi di computer.
È opportuno rilevare che il file oggetto non coincide con il vero e proprio *programma eseguibile*. Ciò è dovuto al fatto che, scrivendo un programma, il programmatore si occupa di esprimere le operazioni che realizzano l'obiettivo che si è prefisso senza tuttavia entrare nel merito di una varietà di servizi necessari per il funzionamento.
Ci riferiamo a servizi del tipo:

- ✓ Stampa di risultati;
- ✓ Memorizzazione o la lettura di dati su file localizzati su disco rigido od altri supporti di memoria;
- ✓ Visualizzazione di immagini;
- ✓ Controllo di dispositivi vari, ecc….

Sono i tipici servizi che il sistema operativo è in grado di fornire e che per essere utilizzati da un programma necessitano di particolari istruzioni o parametri tipicamente raccolti in file che si chiamano **librerie**. La differenza fra file oggetto e librerie sta nel fatto che quest'ultime sono precostituite dal costruttore e vengono fornite all'atto dell'acquisto della licenza e del software di compilazione per un certo linguaggio ed una certa macchina.
Se per esempio dobbiamo scrivere programmi in linguaggio **C** per computer PC con il sistema operativo *Linux*, dovremo avere il compilatore C per PC e sistema Linux, corredato dalle corrispondenti librerie.
Per produrre il programma eseguibile è necessario realizzare un processo di "**collegamento**" fra uno o molti (nel caso di programmi complessi) file oggetto e le opportune librerie, processo che assume il nome di **link**.
Il processo di link è realizzato da un apposito programma che si chiama **linker** e che richiede in ingresso tutti i file oggetto e le librerie necessarie. In uscita il linker produce un file, che si chiama file eseguibile e che contiene il programma in linguaggio macchina che il computer è in grado di eseguire.
Eseguire un programma significa leggere (istruzione per istruzione) dall'hard-disk il file eseguibile che lo contiene, caricarlo nella memoria RAM, la quale provvede in modo rapido a fornire alla CPU le operazioni da eseguire, dati e programmi su cui eseguirle, in una forma codificata che usa simboli dell'alfabeto binario cioè 0 e 1.

Linguaggi interpretati

Anche con i linguaggi interpretati deve avere luogo il processo di traduzione. Tuttavia in questo caso le istruzioni della sorgente non vengono tradotte tutte insieme per produrre un file oggetto come avviene con il compilatore, bensì ogni istruzione viene letta dal file sorgente, ed eseguita subito. In questo caso quindi i processi di traduzione e di esecuzione sono mescolati e non distinti come nel caso dei linguaggi compilati.

In tale descrizione manca evidentemente il concetto di libreria che avevamo introdotto a proposito della compilazione. Nei linguaggi interpretati invece del compilatore abbiamo **l'interprete** che provvede a leggere le istruzioni, a tradurle e ad eseguirle. È l'interprete che esaudisce le richieste di servizi esterni richiesti dal programma. Si può descrivere questa situazione supponendo l'interprete come uno "strato di software" che *avvolge* il programma.

Dal punto di vista della procedura è l'interprete ad essere eseguito e come tale esso incorpora tutte le librerie necessarie all'esecuzione dei programmi. Per essere eseguito, richiede un file sorgente da eseguire; si può affermare che la tecnologia basata sulla compilazione è dominante. Tutti i linguaggi principali utilizzati per la produzione di software di ogni tipo sono compilati.

La compilazione si presta a produrre programmi ottimizzati (l'ottimizzazione di un programma consiste in una serie di accorgimenti che, senza alterare la funzionalità di base, consentono di migliorarne le prestazioni e di ridurre le dimensioni) e che sono distribuiti e commercializzati come pacchetti software indipendenti.

Così sono stati prodotti tutti i pacchetti a cui siamo stati abituati ad utilizzare per scrittura di testi, disegno, ritocco fotografico, video giochi ecc….

La tecnologia basata sull'interpretazione è nata per la realizzazione di linguaggi adatti all'uso didattico, idonei cioè all'apprendimento significativo della programmazione. Nacque così il **Basic** che nella sua forma originale era un interprete: si faceva girare il programma basic (una versione molto diffusa è stata a lungo il **Quick-Basic**) che offriva un *prompt* (per definizione si chiama prompt un segno convenzionale che alcuni programmi offrono affinché l'utente scriva dei comandi, un esempio è il prompt del sistema operativo DOS che usualmente sui sistemi si presenta così: **C:**) al quale si potevano scrivere le singole istruzioni che venivano eseguite immediatamente. Questo sistema viene utilizzato per sperimentare in modo immediato gli effetti delle varie istruzioni o di brevi sequenze di istruzioni. Successivamente il Basic si è evoluto in un linguaggio compilato e completo come molti altri (**VisualBasic**).

La tecnologia basata sull'interpretazione ha avuto tuttavia alcune importanti applicazioni. Una di queste concerne i cosiddetti ambienti di analisi di dati scientifici. Si tratta di programmi che offrono un ambiente con molti comandi che facilitano molto la lettura, manipolazione e visualizzazione di dati). Quasi sempre i vari comandi offerti da tali ambienti possono essere raggruppati in sequenze (si parla in tal caso di **script** che possono essere successivamente eseguite come fossero dei programmi.

In questo modo l'utente può estendere moltissimo la funzionalità dell'ambiente in funzione delle proprie specifiche necessità. Alcuni di questi ambienti consentono anche di tradurre gli script prodotti in veri e propri programmi scritti in un linguaggio da compilare (per esempio C) per produrre dei programmi distribuibili convenzionalmente.

Un'altra evoluzione importantissima della tecnologia d'interpretazione è quella che ha condotto a linguaggi del tipo **Java**.

Livelli di astrazione e macchine virtuali

Macchina virtuale M_n
Linguaggio virtuale L_n
……

Macchina virtuale M_3
Linguaggio virtuale L_3

Macchina virtuale M_2
Linguaggio virtuale L_2

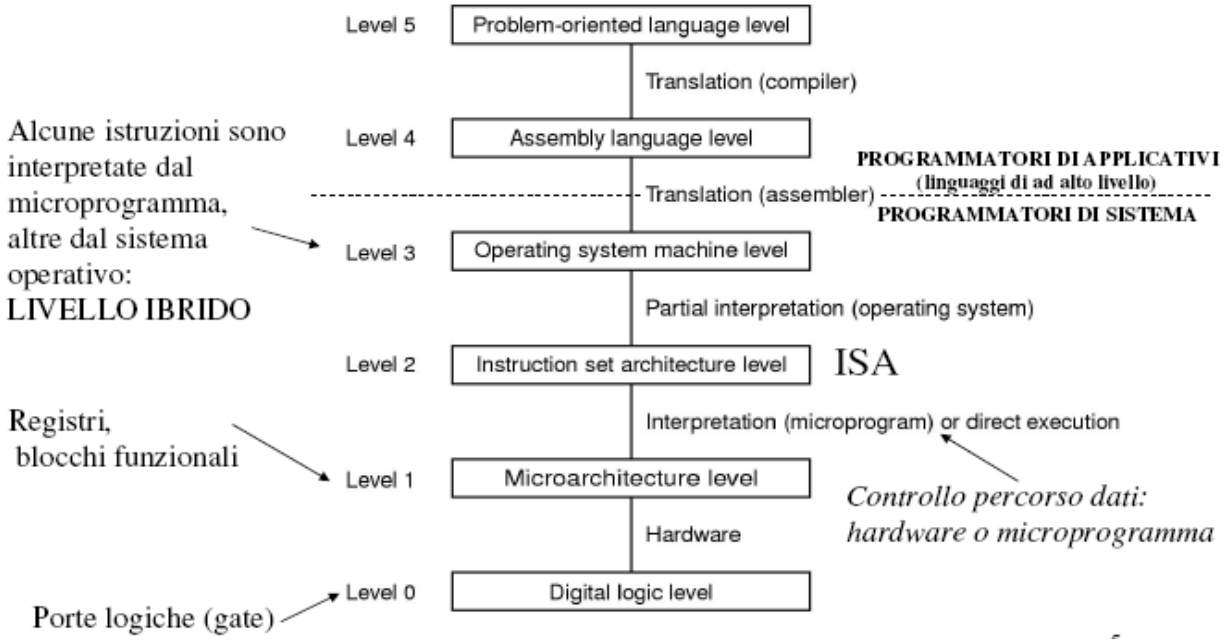
Macchina virtuale M_1
Linguaggio virtuale L_1

Macchina virtuale M_0
Linguaggio virtuale L_0

Dal livello *i* si scala al livello *i-1*

Ogni linguaggio macchina della macchina virtuale sottostante e visto come hardware.

MACCHINA MULTILIVELLO



Livello 5 [livello del linguaggio orientato al tipo di problema]
Livello 4 [livello del linguaggio assemblativo]
Livello 3 [livello macchina del sistema operativo]
Livello 2 [livello ISA]
Livello 1 [livello di micro-architettura]
Livello 0 [livello logico-digitale]

Iniziamo dal livello 0 **[livello logico-digitale]**

- Composto da transistor assemblati in modo da funzionare come dispositivi digitali
- Gli oggetti elementari sono detti porte (gate) e possono assumere solo i valori 0 e 1

Opportune combinazioni di porte consentono di creare memorie di 1 bit (***BI**nary digi**T*** – cifra binaria), corrisponde allo stato di un dispositivo fisico che viene interpretato come **1** o come **0**. I valori 0 e 1 sono rappresentati in modi differenti da diversi dispositivi: da una differente tensione elettrica (alta o bassa) nella memoria centrale, da un differente stato di polarizzazione magnetica (positiva o negativa) nella memoria di massa, dall'alternanza fra luce e oscurità nella trasmissione dei dati. Qualunque siano le motivazioni, la codifica corrisponde a un fenomeno di natura fisica che può essere osservato in due stati distinti.
Di solito le cifre binarie all'interno di un sistema di elaborazione vengono trattate a gruppi o pacchetti contenenti un numero costante di bit: in particolare, per essere elaborate, le cifre binarie vengono raggruppate in sequenze o stringhe di 8 bit. Una stringa di 8 bit, assume il nome di **Byte**.
Dunque il byte, rappresenta un'unità di misura. Un byte può essere utilizzato per generare $\mathbf{2^8}$ differenti sequenze di 1 e di 0 (00000000, 00000001, …, 11111111).
È bene ricordare, che per il trattamento dei dati numerici, gli elaboratori operano su sequenze composte da un numero fisso di byte; Tali stringhe di byte assumono il nome di **parole**.
Esistono elaboratori che operano con parole di:
1 Byte (8 bit) - 2 Byte (16 bit) - 4 Byte (32 bit) - 8 Byte (64 bit)

- Combinazioni di memorie consentono di ottenere I registri a (8,16, 32, 64 bit).

Es. un chip di silicio vi sono decine di migliaia di transitor (μm).

Livello 1 [livello di micro-architettura]
Contiene:

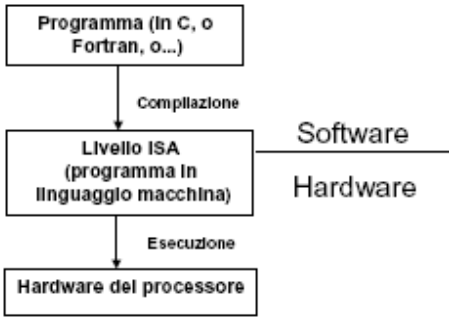
- Un insieme di registri (da 8 a 32)
- Un circuito detto ALU

I registri sono connessi alla ALU tramite il *datapath*, lungo il quale si spostano i dati:

- Selezione di 1 o 2 registri
- Operazione sul loro contenuto
- Memorizzazione del risultato in un registro

Livello 2 [livello ISA – Instruction Set Architecture]
Contiene:

- L'insieme delle istruzioni (Instruction set) che possono essere direttamente interpretate dalla microarchitettura del processore (macchina)
- È il livello cui si fa riferimento quando si descrive il linguaggio macchina di un elaboratore
- È la descrizione dell'elaboratore che serve per scrivere un suo compilatore



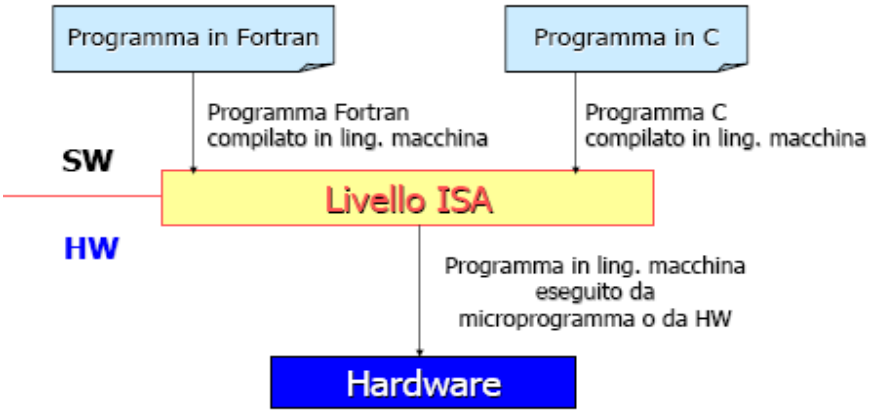
Che cosa descrive l'ISA
L'ISA descrive:

- L'insieme delle istruzioni macchina (formati, operazioni, tipi di dati, tipi di indirizzamento)
- I dati relativi ai programmi
- I riferimenti alle informazioni (indirizzi)
- Le modalità operative
- Gli elementi a disposizione delle istruzioni: il modello della memoria e i registri (sono usabili tramite le istruzioni macchina)

Riferimenti alle informazioni (Specificazione della struttura fisica che supporta il dato)
In linguaggio macchina ci si riferisce ad un dato specificandone la posizione in memoria.
• Un dato può trovarsi in una delle seguenti memorie:
– Memoria centrale (*Riferimento esplicito* all'indirizzo di una locazione di memoria)
– Registro (*Riferimento esplicito* al numero di registro)
– Stack (*Riferimento implicito* ad un registro (stack pointer) che contiene l'address del top dello stack)
Operazioni a livello ISA
Caricamento (da memoria) = trasferimento del contenuto di una parola di memoria in un registro di CPU

- Lettura da memoria:
- Load sorgente (= memoria), destinazione
- MOVE sorgente (= memoria), destinazione
- Memorizzazione (in memoria) = trasferimento del contenuto di un registro di CPU in una parola di memoria
- Scrittura in memoria:
- Store sorgente, destinazione (= memoria)
- MOVE sorgente, destinazione (= memoria)

Es. input di istruzioni → output (in mezzo vi è la "scatola nera" che elabora le istruzioni. L'hardware della 'scatola nera' non è di nostro interesse perché si trova ad un livello inferiore).



Nota
Per definire il livello ISA occorre specificare come il programmatore a livello di **linguaggio macchina** interagisce con il livello HW.
Parametri che definiscono il livello ISA
• **Tipi di istruzioni (**Quante e quali istruzioni sono disponibili)
• **Tipi di dati (**I vari tipi di dati sui quali vengono eseguite le operazioni)
• **Registri (** I registri della CPU referenziabili da parte delle istruzioni, e il loro uso)
• **Modello di memoria (**Quale è la organizzazione della memoria)
• **Riferimento ai dati (**I modi per specificare gli indirizzi degli operandi)

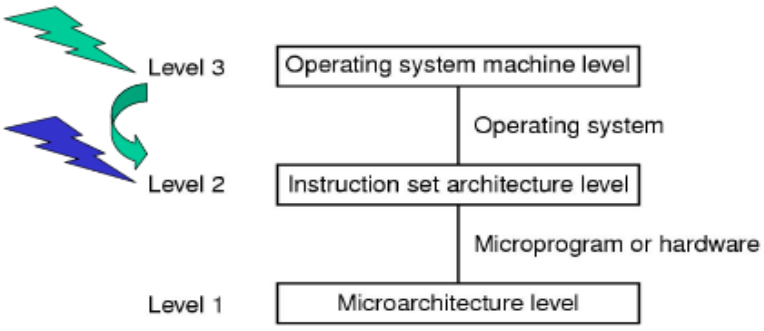
Pertanto per programmi in linguaggio macchina occorre conoscere:
– L'organizzazione della memoria e dei registri
– Il Set di istruzioni macchina disponibili
- Tipi di istruzioni
- Formato di istruzione
– I tipi di dati che è possibile trattare
• Meccanismi di accesso ai dati
• Modalità di indirizzamento degli operandi
L'insieme di queste informazioni definisce l'ISA

Le caratteristiche computazionali di un elaboratore dipendono in gran parte dal livello ISA
Diversi processori sono compatibili tra loro se si basano sulla stessa ISA

Livello 3 [livello macchina del sistema operativo]
Contiene:
- Istruzioni presenti al livello ISA
- Nuove istruzioni
- Organizzazione della memoria
- Gestione concorrente

Nota: I servizi del livello sono compiuti da un interprete (S. Op) eseguito al livello2. (Es. nel sistema operativo Unix siamo al livello3).

- Il livello macchina del sistema operativo (OSM) contiene tutte le istruzioni disponibili ai programmatori, pressoché tutte le istruzioni del livello ISA, e le nuove istruzioni aggiunte dal sistema operativo (chiamate di sistema - *system call*).
- Una *system call* invoca un predefinito servizio del sistema operativo (es. lettura/scrittura da/in un file).
- Il livello OSM è interpretato, quindi "dietro" la chiamata di sistema vi è un'*utility* che esegue il servizio a livello ISA.
- Le chiamate fatte dal programma a livello ISA "non transitano" per il sistema operativo.



Livello 4 [livello del linguaggio assemblativo]
Consente di scrivere un programma in maniera più semplice e meno complessa rispetto ai livelli sottostanti. Il programma che esegue la traduzione viene detto assemblatore dei linguaggi.

- Le istruzioni ISA sono in corrispondenza uno a uno con quelle del linguaggio Assemblatore. Nella descrizione del livello ISA si farà riferimento alla notazione simbolica delle istruzioni e delle variabili

Il *linguaggio assemblativo* (*assembly language*) rappresenta il primo passo verso la semplificazione della programmazione.

- Il linguaggio Assemblatore è il linguaggio *simbolico* che consente di programmare un elaboratore utilizzando le istruzioni del linguaggio macchina.
- L'operatore esprime il proprio programma in linguaggio assemblativo, dopodiché un programma (l'assemblatore) traduce le istruzioni in linguaggio macchina, producendo così un programma che è equivalente al primo, ma espresso in una forma utilizzabile direttamente dalla macchina reale.

I compiti dell'assemblatore sono:
- Assegnare automaticamente gli indirizzi a dati e istruzioni
- Tradurre i nomi simbolici in indirizzi e valori
- Trasformare le istruzioni dalla forma simbolica a quella binaria.

- Per poter essere eseguito, un programma scritto in ASSEMBLER deve essere tradotto in linguaggio macchina, in modo tale da tradurre i codici mnemonici delle istruzioni in codici operativi, sostituire tutti i riferimenti simbolici degli indirizzi con la loro forma binaria, e riservare lo spazio di memoria per le variabili.
- Se nel codice sorgente sono presenti riferimenti simbolici definiti in moduli esterni a quello assemblato è necessario anche il LINKER (collegatore).

L'assemblatore è un particolare tipo di programma per i quale i dati di ingresso e di uscita sono altri programmi.

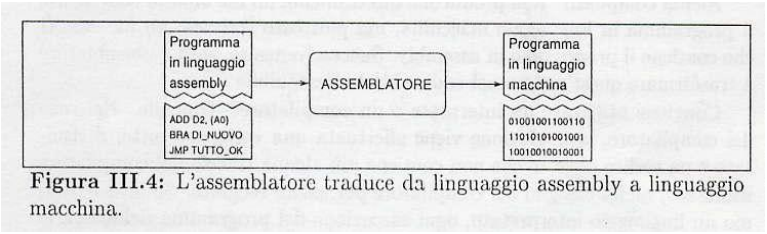


Figura III.4: L'assemblatore traduce da linguaggio assembly a linguaggio macchina.

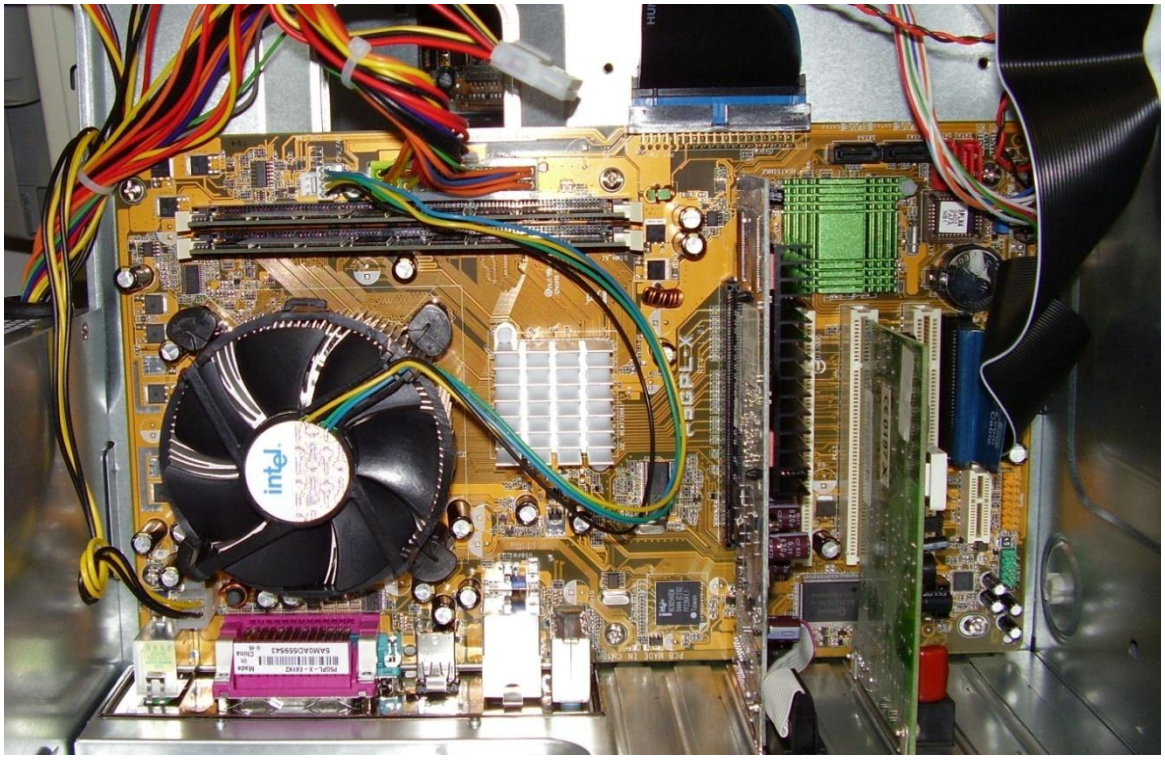
I programmi che, come l'assemblatore, effettuano la traduzione da un linguaggio a un altro sono detti *traduttori*.
Il linguaggio assembly, come il linguaggio macchina, riflette l'architettura interna del processore utilizzato.

Quindi:
- ogni CPU ha il proprio linguaggio assemblativo
- programmare in linguaggio assemblativo implica la conoscenza della struttura logico-circuitale del processore target (numero di registri, spazio degli indirizzi...)
-un programma scritto il linguaggio assemblativo può essere impiegato sul processore target.

Livello 5 [livello del linguaggio orientato al tipo di problema]
Consiste in linguaggi utilizzati per realizzare interfacce grafiche o applicazioni.

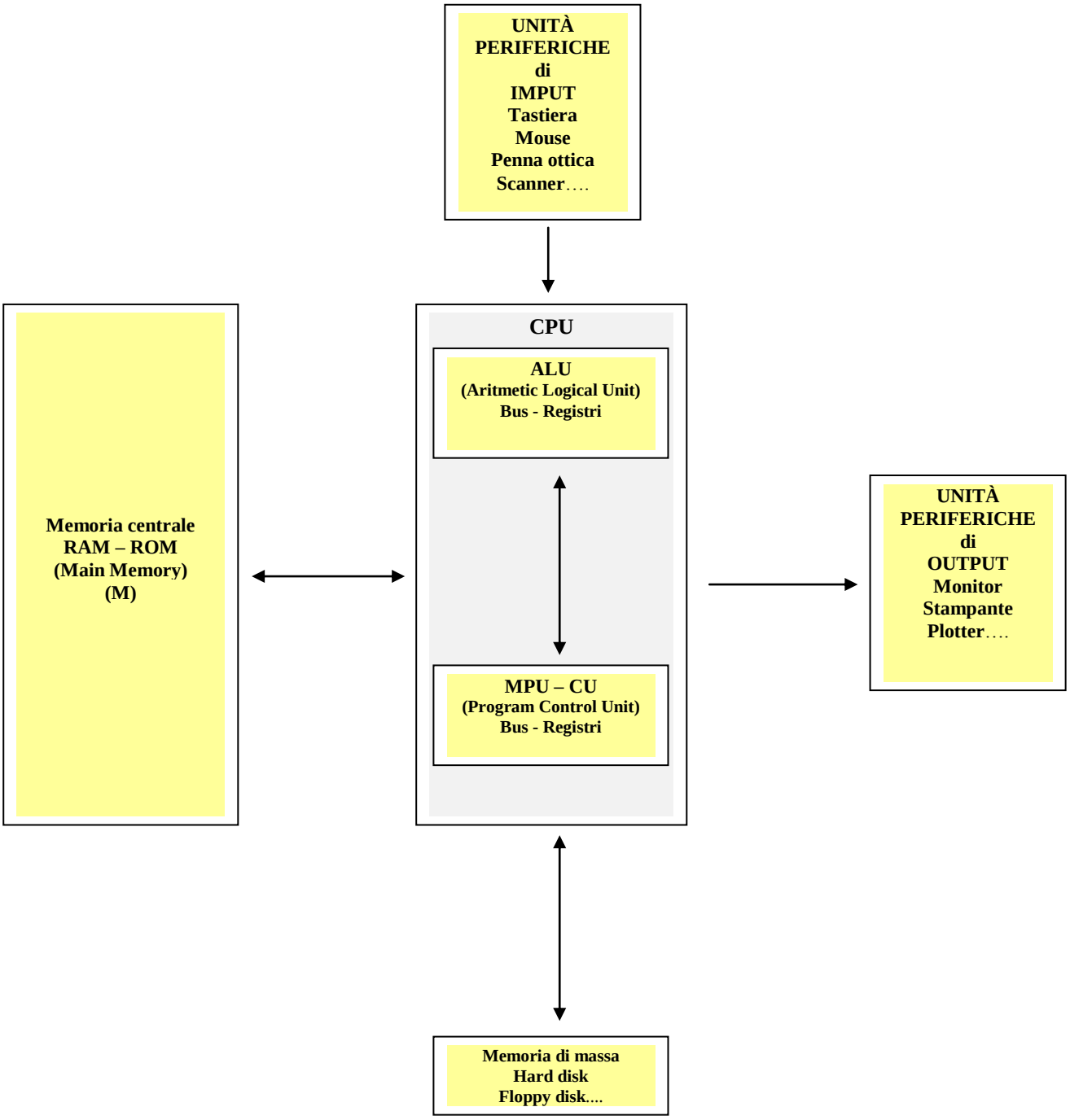
Elementi funzionali della macchina di von Neumann

La macchina di Von Neumann è costituita da quattro unità funzionali fondamentali: **l'unità di elaborazione centrale o processore** (CPU, *Central Processing unit*), la **memoria centrale** (RAM, *Random Access Memory*), le **periferiche I/O** e il **bus dell'elaboratore**. Come già accennato, questi componenti hardware sono stati definiti nel paragrafo 1.5.1.



L'unità di elaborazione contiene i dispositivi elettronici in gradi di acquisire, ed eseguire le istruzioni del programma. La memoria centrale contiene le informazioni necessarie all'esecuzione di un programma, cioè istruzioni e dati. Le periferiche permettono lo scambio delle informazioni fra l'elaboratore e il mondo esterno, attraverso operazioni di imput – dall'esterno all'elaboratore – e di output – dall'elaboratore all'ambiente esterno -. In particolare, sono parti integranti dell'elaboratore le sole **interfacce** di collegamento verso le periferiche, mentre queste ultime vengono considerate come dispositivi separati. Nel tipo di architettura considerata, i dispositivi **I/O** includono anche le memorie di massa. Infine, il bus dell'elaboratore opera il collegamento fra queste unità funzionali.

Schema semplificato di un elaboratore



Osservazione
Da notare che questa architettura elementare è un'astrazione della macchina reale, in cui sono presenti ulteriori componenti; tuttavia, tale astrazione è in prima approssimazione vicina alla realtà in modo tale di fornire uno schema del funzionamento di un calcolatore.

Il funzionamento della macchina può essere descritto nel modo seguente:
L'unità di elaborazione coordina le varie attività, nel caso specifico esamina istruzioni dalla memoria centrale, decodifica il loro significato e le esegue per mezzo di opportune funzioni all'interno dell'architettura del calcolatore. Le istruzioni possono comportare attività di elaborazione dei dati (per esempio, calcoli numerici) oppure attività di trasferimento dei dati (per esempio, dall'interfaccia di una periferica alla memoria centrale). I trasferimenti tra elementi funzionali diversi avvengono tramite il bus di sistema che, in funzione dell'operazione del momento, effettua il collegamento logico tra i vari elementi funzionali coinvolti nel trasferimento. Le fasi di elaborazione sono sincronizzate rispetto alla scansione temporale imposta da un **orologio di sistema** (*system clock*). Durante ciascun intervallo di tempo, l'elemento circuitale detto unità di controllo coordina l'esecuzione temporale delle funzioni che verranno svolte all'unità di elaborazione stessa o negli altri elementi elettronici.

Hardware
L'hardware di un elaboratore è costituito da un insieme di componenti elettronici, atti ad assolvere la complessa funzione di elaborazione. I componenti più importanti sono:
L'unità di elaborazione centrale, o **processore** (**CPU**, acronimo di *Central Processing Unit*), svolge le elaborazioni e coordina il trasferimento dei dati all'interno del sistema informatico. Il processore ha il compito di eseguire le varie istruzioni da cui i programmi sono composti.
Gli elementi circuitali che costituiscono la CPU sono:

- ☑ **L'unità di microelaborazione** (**MPU** acronimo di *Microprocessor unit*), che esegue le funzioni centrali di controllo di un computer, con i bus dati, degli indirizzi e dei controlli che sono le linee attraverso le quali viene trasferita l'informazione tipo ai vari dispositivi del sistema (stampante, monitor, ecc)
- ☑ **L'unità di controllo** (**CU**, acronimo di *Control Unit*) è responsabile del prelievo e della decodifica delle istruzioni nonché dell'invio dei segnali di controllo che provocano i trasferimenti o le elaborazioni necessarie per l'esecuzione.
- ☑ **L'unità aritmetico-logica** (**ALU**, acronimo di *Aritmetic and Logic Unit*) realizza le operazioni aritmetiche (+ - * / ^) e logiche AND – OR – NOT) richieste per l'esecuzione dell'istruzione.
- ☑ **Bus** (Il bus dell'elaboratore è l'organo destinato a collegare le varie unità funzionali (processore, la memoria e le varie interfacce I/O) che lo compongono.
- ☑ **L'orologio di sistema** (*clock*) sincronizza le operazioni rispetto ad una determinata frequenza.

Approfondimenti
L'unità di controllo fornisce all'unità di elaborazione i segnali elettrici che attivano i diversi dispositivi di memoria o di operazione. Questi segnali vengono forniti in sincrono con un orologio interno delle macchina, come già detto clock: ad ogni scatto dell'orologio di sistema viene trasmesso un segnale (**Clock**: la frequenza di funzionamento del processore).
La frequenza con cui l'orologio di sistema scatta fornisce una precisa indicazione sulla velocità attraverso la quale opera la CPU: la frequenza è misurata in megahertz (**MHz**) o (**GHz**), milioni o miliardi di cicli al secondo; L'unità di misura della velocità di un'unità centrale è il **MIPS** (Millions Instructions al secondo).
La CPU contiene elementi di memoria che possono avere diverse funzioni, e tra essi si distinguono:

- ❖ **Memorie** tipo ROM, contenenti informazioni permanenti del dispositivo;
- ❖ **Registri**, sono locazioni (celle) di memoria nelle quali si può leggere e scrivere molto velocemente, finalizzati per memorizzare risultati parziali delle operazioni, informazioni necessarie al controllo, operandi, nonché i codici operativi del linguaggio macchina;

I principali registri della CPU sono:

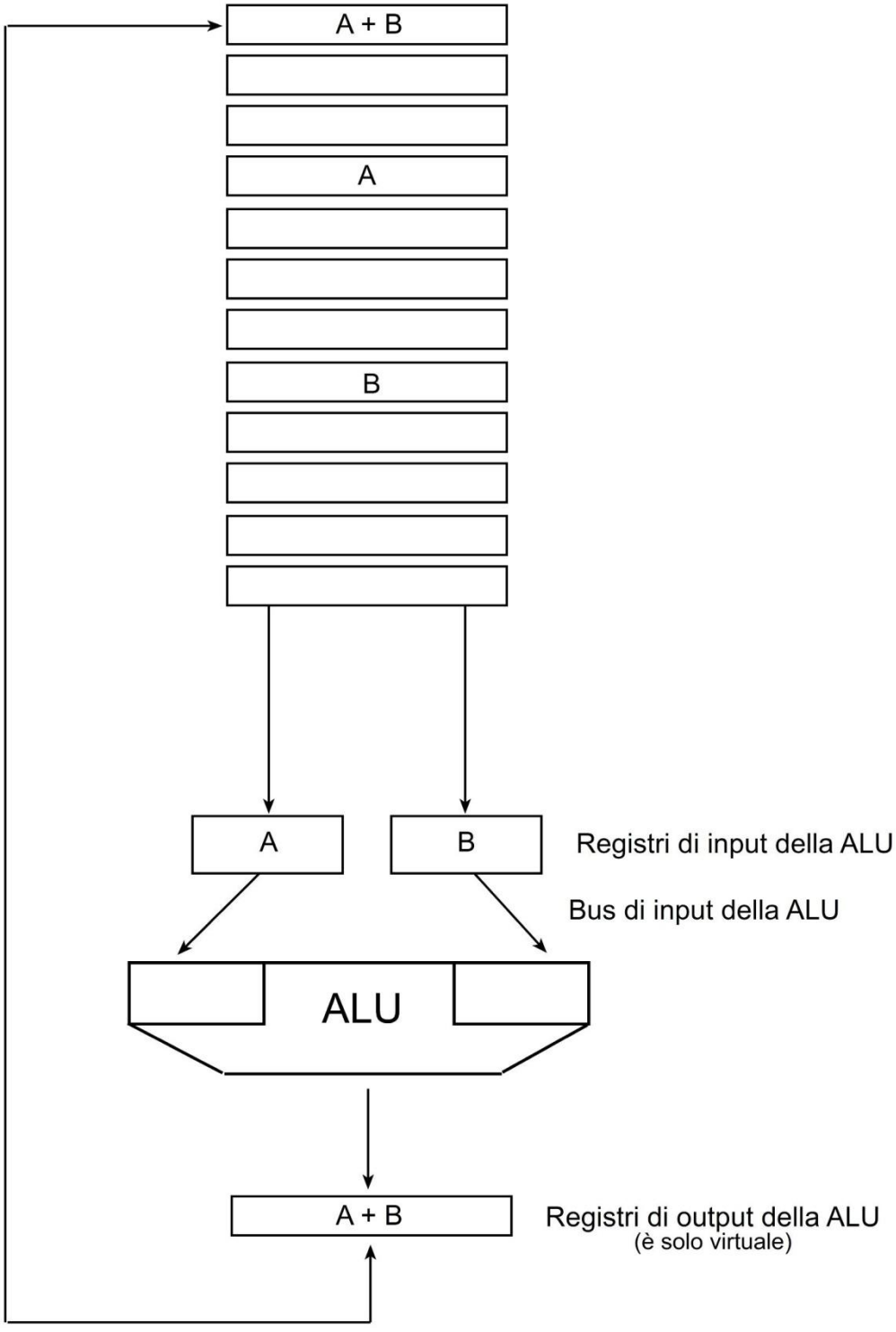
- ❖ **Program Counter**, contenente l'indirizzo della memoria dove si trova la prossima istruzione da eseguire
- ❖ **Il registro istruzione corrente** (CIR, *Current Instruction Register*, lungo *h* bit) che contiene, istante per istante, l'istruzione che risulta attualmente in esecuzione da parte dell'elaboratore.
- ❖ **Registro di stato**, per il controllo delle condizioni o delle anomalie che si verificano durante le operazioni
- ❖ **L'accumulatore**, utilizzato per effettuare calcoli matematici semplici e memorizzare i risultati parziali delle operazioni
- ❖ Il **registro dati di memoria** (DR, *Data Register*, lungo h bit, ossia è pari ad una lunghezza di una parola), contenente il dato prelevato da una cella di memoria o pronto per essere trasferito in un'altra cella.
- ❖ Il **registro indirizzi di memoria** (AR, *Address Register*, lungo *k* bit), contenente l'indirizzo della cella di memoria dove si deve prelevare o depositare un dato o un'istruzione.
- ❖ Un **registro dati dei dispositivi I/O** (PDR, *Peripherical Data Register*), per scambiare dati con la periferica. Lo scambio può avvenire verso la periferica (come ad esempio la stampante) oppure verso la CPU (per esempio con i lettori ottici).
- ❖ Un **registro comando della periferica** (PCR, *Peripherical Command Register*), che contiene il comando che la periferica stessa dovrà eseguire.

Anche nell'unità aritmetico-logica vi sono due registri principali che sono chiamati: **registro A** e **registro B**. Questi due registri sono impiegati per memorizzare gli *operandi* e i risultati delle operazioni.

Sono registri specifici i seguenti:
Registri non operazionali (detti anche **dedicati)**, cioè quei registri che non possono essere manipolati dal programmatore.
PC (Program Counter), IR (Instruction Register), MAR (Memory Address Register), PSW (Program Status Word)……
Mentre i **Registri operazionali (**detti anche **generali)** cioè quei registri che possono essere controllati dal programmatore per conservare e manipolare dati (operandi e risultati).
• Tutte le operazioni sono eseguite sui dati contenuti nei registri interni al processore.

- Registro – memoria
  Le parole di memoria sono prelevate dalla memoria e trasferite nei registri.
- Registro
  Gli operandi sono prelevati dai registri e trasferiti nei registri di input delle ALU, utilizzati per una precisa operazione; il risultato viene memorizzato in un apposito registro.

Esempio



Insiemi di istruzioni
Tipologia delle istruzioni
Formato delle istruzioni
Modalità di indirizzamento

Set di istruzioni
- Parte visibile al programmatore e al progettista di compilatori.
- **Istruzione (**Parola del linguaggio macchina)
- **Insieme delle istruzioni (**Vocabolario del linguaggio macchina)
- **Formato di istruzione (**Sintassi della parola)

Formato delle istruzioni

- Il formato di istruzione si riferisce al **modo in cui è rappresentata una istruzione** e sono codificati il codice operativo e gli operandi.
- Il formato di istruzione specifica: operazione da eseguire, numero di operandi, tipo e dimensione degli operandi, collocazione degli operandi.

Lunghezza di istruzione
• Scelta architetturale condizionata dalla dimensione ed organizzazione della memoria, dalla struttura del bus, dalla complessità della CPU.
• Istruzioni brevi rispetto alla larghezza di banda della memoria
• Minore occupazione di memoria
• Maggiore velocità di trasferimento e quindi di esecuzione delle istruzioni
• Difficile la decodifica e la esecuzione concorrente
• Istruzioni lunghe rispetto alla larghezza di banda della memoria
• Maggior occupazione di memoria
• Complessità maggiore per l'unità di controllo
• Tempo di decodifica assente

Introduzione alla Memoria centrale

La **memoria centrale** (**RAM**, acronimo di *Random Access Memory*), utilizzata per memorizzare dati e programmi utili al funzionamento dell'elaboratore. Provvede in modo rapido a fornire alla CPU le operazioni da eseguire, dati e programmi su cui eseguirle, in una forma codificata che usa simboli dell'alfabeto binario cioè 0 e 1.
La memoria centrale in linea generale è caratterizzata dai seguenti parametri:

- **Accesso casuale** (molto veloce);
- Possiede **capacità limitata** e contiene perciò una quantità ridotta di programmi;
- È **volatile**, cioè il suo contenuto viene perduto quando il calcolatore si spegne, oppure qualora vi sia un guasto o un'interruzione di energia elettrica.

Il termine memoria ad accesso casuale o diretto (RAM) si riferisce al fatto che si può accedere direttamente a qualsiasi locazione di memoria, scelta a caso, impiegando lo stesso tempo di accesso, in contrasto con il concetto di memoria ad accesso *sequenziale* dove per raggiungere una parola è necessario leggere tutte le precedenti.
La RAM può considerarsi come divisa in tre aree:

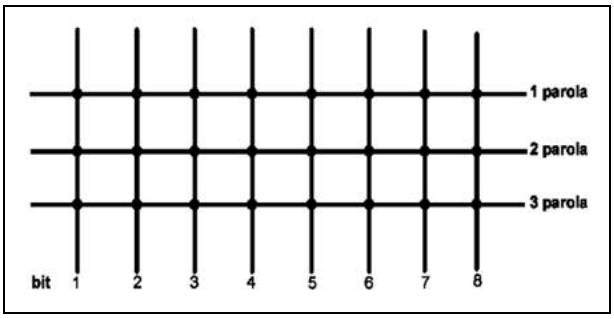
- La prima aerea utilizzata per caricarvi il *nucleo* (*kernel*) *del sistema operativo*;
- La seconda aerea utilizzata per caricarvi il *programma applicativo*;
- La terza aerea è riservata per immagazzinare dati da elaborare e i risultati ottenuti dall'esecuzione.

Approfondimenti
Dal punto di vista concettuale la *memoria centrale* si può assimilare come un *insieme di celle o locazioni di memoria*, ciascuna delle quali è in grado di memorizzare dati e istruzioni.
Ad ogni cella viene assegnato un **indirizzo** (*address*), corrispondente ad un numero d'ordine (a partire da zero), cioè è il simbolo che identifica in modo univoco una sola locazione di memoria, per potervi accedere senza essere confusa con le altre.
Pertanto ogni cella di memoria può essere *indirizzata* ovvero si indica la *capacità dell'elaboratore di selezionare una particolare cella (locazione) di memoria*.
L'indirizzamento della memoria avviene tramite un opportuno registro, detto **Registro Indirizzi** (dispositivo incorporato nella CPU).
Gli indirizzi di memoria sono gli oggetti sui quali un elaboratore deve lavorare: se per esempio un programma richiede di recuperare un dato memorizzato, deve specificare l'indirizzo in cui è contenuto, e per far ciò deve ricordare anche l'indirizzo che gli serve registrandolo in una locazione di memoria specifica.
È preferibile che un indirizzo sia un numero binario, e che sia memorizzabile in uno spazio delle dimensioni di uno o più byte. Il numero di bit utilizzati dalla CPU per identificare gli indirizzi definisce lo **spazio di indirizzamento** (*addressing*).
Osservazione
I registri sono molto veloci della memoria centrale.

Esempio
Supponiamo che il Registro Indirizzi sia lungo 32 bit, posso indirizzare 2^{32} celle diverse.
2^{32} celle = 4 Gigacelle → 4 Gbyte

Osservazione
In generale, un registro è un dispositivo elettronico costituito da sequenze di celle di memoria, capace di memorizzare una sequenza di bit.

Le sequenze di locazioni di memoria sono organizzate in configurazioni multiple di bit, dette **parole** (word).
Ogni parola di memoria è una struttura fisica nella quale è possibile memorizzare uno o più byte, ovvero numeri da 0 a 255. Le più tipiche lunghezze di parola includono i primi multipli del Byte (8bit), e si hanno dunque calcolatori con parole di 8, 16, 32 e 64 bit.
La memoria centrale si può schematizzare come una grande tabella, che ha per righe le varie celle; le colonne, in numero pari alla lunghezza di parola, individuano ciascun bit di memoria.
L'informazione è presente in memoria come stato (alto o basso) di tensione individuata nei nodi dalle righe con le colonne.
Assumiamo che il valore basso di tensione sia associato al valore **0** e il valore alto sia associato al valore **1**.



In definitiva, i parametri che caratterizzano la memoria centrale sono:
- La *dimensione complessiva della memoria* (*capacità*);
- La *modalità di accesso* (solo lettura o anche scrittura);
- La *volatilità*;
- La *velocità con cui risponde alle richieste*.

Osservazione
Per praticità si utilizza come spazi di indirizzamento le potenze del numero 2, data l'intrinseca natura binaria del sistema di numerazione.

Esempio
Vogliamo rappresentare la memoria vista come una sequenza di 65536 celle.
Per indirizzarle occorre un numero da 0 a 65535, e per scrivere tutti gli indirizzi occorrono 16 bit, ovvero 2 byte;
Infatti con 2 Byte si possono indirizzare $2^{16} = 65536$ celle.
Così come con ***n*** bit di indirizzamento si possono gestire memorie di 2^n celle.

I bit contenuti nei registri possono essere interpretati come numeri naturali. Se il registro indirizzi ha k bit, si possono indirizzare fino a 2^k celle di memoria, i cui indirizzi varieranno fra 0 e $2^k - 1$. Questo schema indica il motivo per cui anche la dimensione della memoria, cioè il numero di parole disponibili, è in genere una potenza del 2.
Quando si afferma che il registro indirizzi ha dimensione 10 bit, vuol dire che vengono indirizzate $2^{10} = 1024$ celle; questa dimensione assume il nome di "*kilo-parola*" (in genere la locuzione *kilo* viene associato alla potenza 2^{10}), analogo discorso se il registro indirizzi ha dimensione 20 bit, vuol dire che vengono indirizzate $2^{20} = 1048576$ celle; questa dimensione assume il nome di "*mega-parola*".

Operazioni di lettura e scrittura

Per compiere operazioni di **lettura dalla memoria**, o **scrittura in memoria** occorre selezionare una particolare cella di memoria caricando il registro indirizzi con una sequenza di 0 e 1, che indichi la posizione relativa della cella nella memoria.
Inoltre le suddette operazioni utilizzano un secondo registro presente nell'unità di elaborazione, detto **registro dati**, che ha lunghezza pari ad una parola di memoria.
Pertanto:

☑ L'operazione di lettura attiva la copia del contenuto della cella di memoria nel registro dati
(l'operazione **carica** (load) il registro dati con una parola di memoria).

☑ L'operazione di scrittura copia il contenuto del registro dati in una cella di memoria
(l'operazione **deposita** (store) il contenuto del registro dati in una parola di memoria).

In altre parole, si può affermare la memoria può essere letta e scritta per effetto delle istruzioni di un programma; e come già detto, si tratta di una memoria ad accesso casuale, cioè consente di scegliere una qualsiasi cella di memoria per una operazione di lettura e scrittura.

Introduzione al Ciclo: fetch – decode – execute

Esecuzione dell'operazione

1) Prelievo dell'istruzione successive dalla memoria e scrittura in IR
2) Modifica del program counter (aggiornamento)
3) Determinazione del tipo di istruzione
4) Locazione della parola (se in memoria centrale)
5) Prelievo della (se in memoria) e trasferirla in un registro
6) Esecuzione dell'istruzione
7) Ritorno a 1

Istruzioni

Due modi di eseguire le istruzioni:

- Realizzare un processore hardware che esegua le istruzioni
  (componenti hardware talvolta complessi: Porte logiche)
- Scrivere un programma capace di interpretare tali istruzioni
  L'interprete scompone le istruzioni in istruzioni più semplici (dette *microistruzioni*), quindi il processore che le esegue può essere molto semplice ed economico.

CISC vs RISC

Distinguiamo due filosofie:

- Istruzioni sempre più complesse (e numerose)
- Istruzioni più semplici (e meno numerose)

Nei primi anni 80:

- Processori RISC (Reduced Instruction Set Computer)
  Sviluppati da IBM (serie RS6000)
- Processori MIPS

Caratteristiche:

- Poche istruzioni semplici
- Avvio veloce delle istruzioni
- Mancanza di interprete
  (ridurre I costi per leggere i dati sui due registri – es. di lettura di due registri nell'addizione)

Principi di progettazione degli elaboratori

- Tutte le istruzioni sono eseguite direttamente dall'hardware
  (evitare l'interpretazione)
- Massimizzare la frequenza di avvio delle istruzioni
  (cercare di avviare il maggior numero di istruzione nell'unità di tempo)
- Massimizzare i riferimenti della memoria, e il numero di registri possibili

parallelismo