I.S.I.S. Raffaele del Rosso - Giovanni da Verrazzano

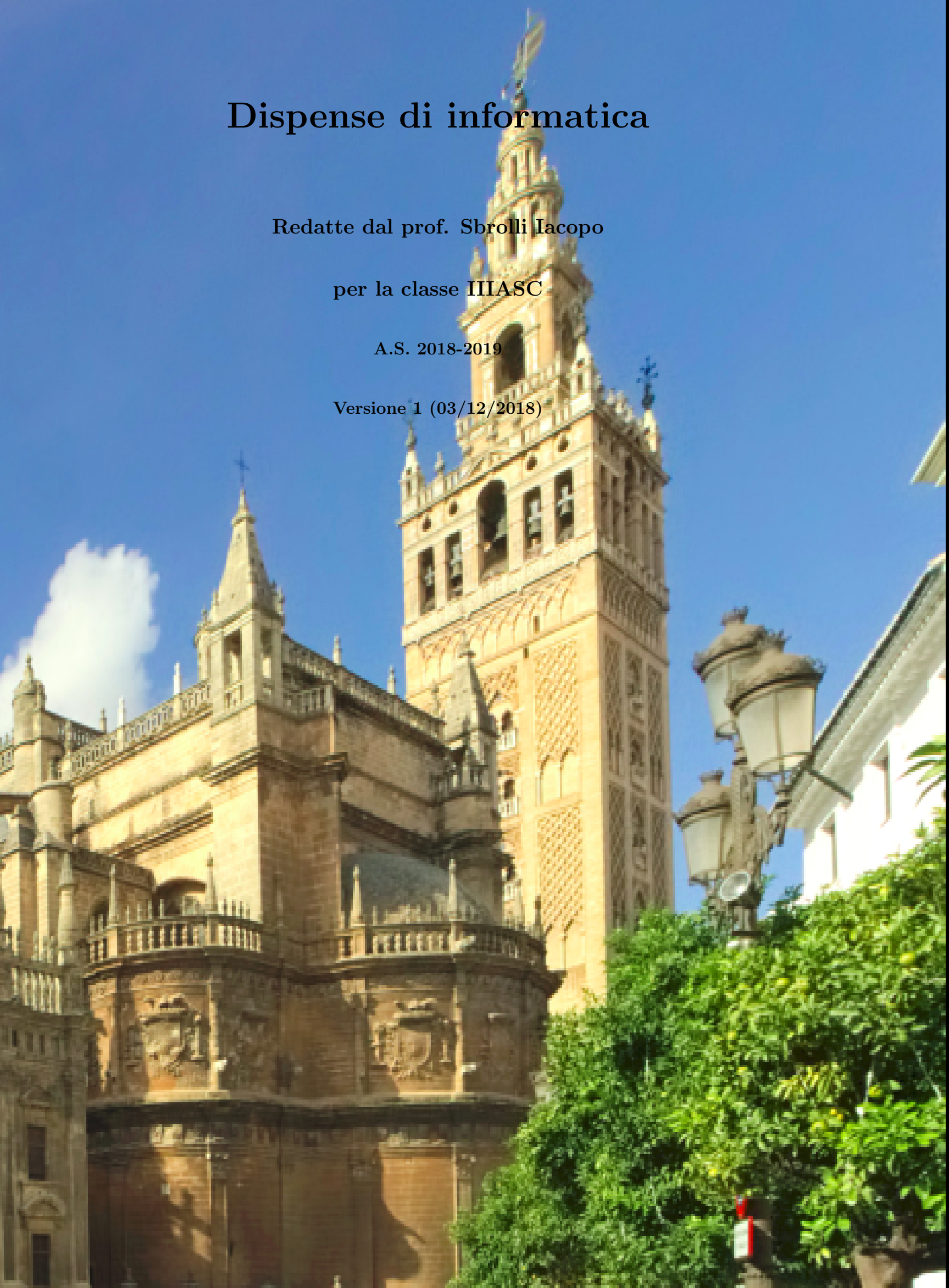
Dispense di informatica

Redatte dal prof. Sbrolli Iacopo

per la classe IIIASC

A.S. 2018-2019

Versione 1 (03/12/2018)



Indice

Sezione 1

Nozioni base di informatica

L'**informatica** è la scienza che studia le metodologie di gestione delle informazioni. Le metodologie di gestione di cui si occupa l'informatica non sono generalmente attuabili dalla mente di un essere umano, a causa della loro complessità. Esse sono concepite e formulate per essere messe in atto da un *calcolatore elettronico*.

Il **calcolatore elettronico** è una macchina complessa composta da minuscoli circuiti utilizzata principalmente per *leggere*, *elaborare* e *immagazzinare* informazioni di qualsiasi tipo. Tali informazioni sono elaborate e immagazzinate tramite supporti fisici (*hardware*) o virtuali (*software*).

Esiste una differenza sostanziale tra **lettura**, **elaborazione** e **immagazzinamento** dell'informazione.

La **lettura** è un'operazione di decodificazione che permette al computer di incamerare dati provenienti dall'esterno. L'**elaborazione** è rappresentata da un insieme di **processi** che utilizzano i dati iniziali forniti al computer tramite la **lettura** (detti *input*) per produrre i risultati desiderati (detti *output*). L'**immagazzinamento** è rappresentato dallo stoccaggio degli *output* in una **unità di memoria**.

In sintesi, l'informatica studia metodi ideali per *leggere* insiemi di dati, *elaborarli* per ottenere un risultato specifico e poi di *immagazzinare* i prodotti.

1.1 Hardware

L'*hardware* è il supporto fisico tramite cui il computer funziona. Il mouse, la tastiera, lo schermo, la CPU, la scheda video e il disco rigido sono componenti dell'*hardware*.

Le componenti dell'*hardware* svolgono essenzialmente tre funzioni:

1. **Lettura** degli *input*. Questo compito è svolto ad esempio dalle cosiddette *periferiche*, come la tastiera e il mouse. Infatti, tramite questi dispositivi possiamo fornire al computer i dati da elaborare.

2. **Elaborazione** degli *input* e produzione degli *output*. Questo compito è svolto dalla CPU, l'unità di **elaborazione** centrale.

3. **Immagazzinamento** o visione degli *output*. Questa operazione viene svolta dalla memoria di massa (il disco rigido) e dallo schermo (che ci mostra i risultati delle operazioni svolte).

Tutti questi dispositivi sono legati dal cosiddetto *bus di sistema*, che trasferisce i dati da una componente all'altra del calcolatore.

1.1.1 Lettura

Dal punto di vista dell'*hardware*, la **lettura** delle informazioni viene effettuata tramite periferiche come la tastiera e il mouse. Le informazioni vengono lette **anche dalle varie tipologie di memoria del computer**: esistono alcuni tipi di memoria *a sola **lettura*** (come la ROM, acronimo per *read only memory*). Tramite la tastiera e il mouse possiamo infatti inviare degli impulsi alla unità di **elaborazione** centrale (la CPU). Se ad esempio premiamo sul tasto «x» che troviamo nell'angolo in alto a destra delle finestre dei programmi, invieremo un segnale alla CPU chiedendole di eliminare dalla memoria RAM le informazioni relative a quel programma.

La ROM è un tipo di memoria molto particolare. Essa, infatti, contiene informazioni basilari (come quelle necessarie all'accensione del computer) che solitamente non vengono mai modificate o cancellate.

1.1.2 Elaborazione

Come abbiamo già detto in precedenza, l'**elaborazione** delle informazioni è gestita dalla **CPU** (*central processing unit*, in italiano **unità di elaborazione centrale**). **La CPU è il cuore del calcolatore**: essa effettua le operazioni logiche e aritmetiche elementari (come la somma) che costituiscono l'impalcatura di qualsiasi algoritmo. Essa è composta da tre elementi fondamentali:

1. l'**unità di controllo** (CU, ovvero *control unit*), il cui scopo è la gestione dei dati in ingresso e in uscita dalla CPU.

2. l'**unità aritmetica centrale** (ALU, ovvero *arithmetic central unit*) la quale effettua le operazioni aritmetiche (somme, sottrazioni) e logiche (negazione, congiunzione logica, disgiunzione logica) necessarie per il funzionamento del computer

3. l'**unità di gestione della memoria** (MMU, ovvero *memory management unit*), il cui scopo è la gestione della *memoria* di cui il processore necessita per salvare i dati che produce.

Le performance della CPU dipendono dalla cosiddetta **frequenza di clock**, ovvero dal numero di operazioni che la CPU può effettuare al secondo. I moderni processori intel i3, i5 e i7 (presenti nella maggior parte dei computer che utilizzate) hanno una frequenza di clock compresa tra i 2,4 e 4,2 gigahertz, ovvero possono effettuare tra i 2,4 e i 4,2 miliardi di operazioni al secondo.

La CPU legge i comandi codificati in codice binario. L'*architettura* della CPU fornisce il numero di bit che compongono le **parole** che l'unità di **elaborazione** può leggere. Ad esempio, se l'architettura della CPU è a 32 bit, la CPU potrà interpretare 2^{32} **parole** (circa 4 miliardi).

1.1.3 Immagazzinamento

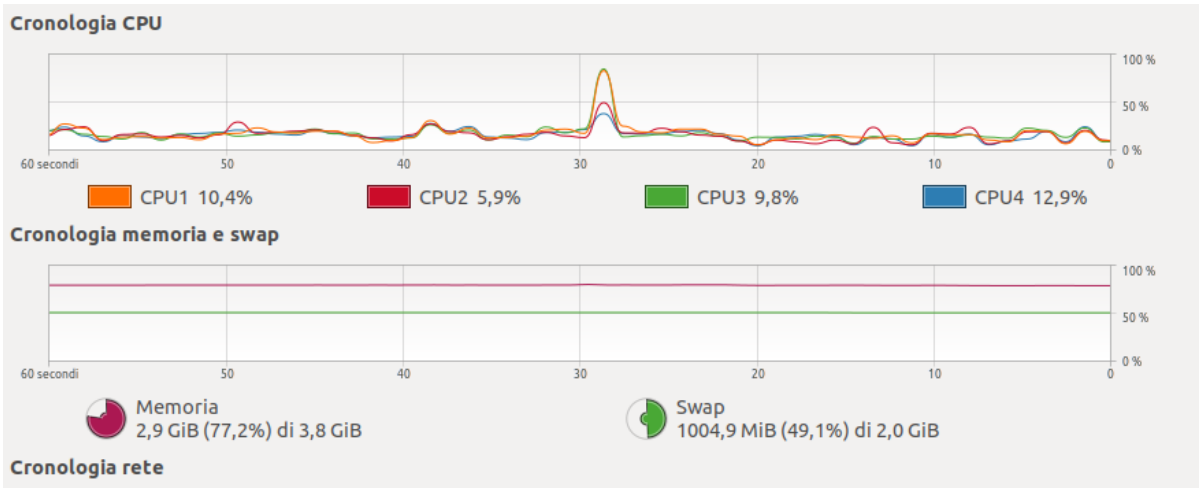
L'**immagazzinamento** delle informazioni prodotte dalla CPU avviene essenzialmente tramite tre supporti: **la memoria cache**, **la RAM** e la **memoria di massa**. La memoria cache e la RAM si somigliano tra di loro: esse infatti contengono informazioni **temporanee**, il cui scopo è la gestione dei processi in atto (i programmi aperti: ad esempio, se aprite *paint* o *word* occuperete la cache e la RAM). La memoria di massa, invece, serve a immagazzinare le informazioni in modo permanente.

Vediamo più nel dettaglio una panoramica dei tipi di memoria del computer.

La memoria cache è strettamente collegata alla CPU, ed è molto più piccola della RAM (da qualche kilobyte a qualche megabyte). Essa contiene solo le informazioni che vengono utilizzate più frequentemente dalla CPU, e serve a ottimizzare il lavoro dell'elaboratore.

La RAM contiene gran parte delle informazioni temporanee gestite dalla CPU (i programmi). Nei moderni computer, la RAM contiene solitamente 4 o più gigabyte di informazioni (si veda la figura 1.1). Se il computer viene spento, la RAM si svuota. La RAM è composta da piccoli **indirizzi** di memoria, a ciascuno dei quali è associato byte di informazione (8 bit).

La memoria di massa (disco rigido) contiene le informazioni che l'utente desidera salvare permanentemente. Essa può avere una dimensione che supera il terabyte nei moderni computer. **Tramite la MMU**, la CPU può utilizzare parte della memoria di massa per salvare dati temporanei (come quelli della RAM). La parte di disco rigido utilizzata a tale scopo è detto *memoria swap*. In pratica, **la CPU utiliza la memoria di massa del disco rigido per tenere aperti i programmi usati molto raramente**, in modo tale che la RAM sia tenuta libera. Tuttavia, la *memoria swap* non può essere considerata un sostituto della RAM, perché essa comunica le informazioni che contiene molto lentamente alla CPU (circa 1000 volte più lentamente della RAM). **Se il vostro computer si blocca, sta probabilmente utilizzando la *memoria swap* e la RAM è piena**.



(a)

Nome del processo	Utente	% CPU	ID	Memoria
Web Content	iacopo	0	10868	398,5 MiB
Web Content	iacopo	1	19071	396,6 MiB
firefox	iacopo	0	2423	257,2 MiB

(b)

Figura 1.1: Pannello (a): memoria RAM, memoria swap e CPU utilizzate dal mio calcolatore il 3 dicembre 2018. Come potete vedere, il mio computer ha una RAM di circa 3,8 gibibyte, di cui 2,9 risultano occupati. La memoria swap totale (ovvero la porzione di disco rigido utilizzato come RAM ≪virtuale≫) è pari circa 2 gibibyte, di cui 1 è occupato. Nel computer sono presenti 4 CPU. Le percentuali che compaiono accanto a ciascuna CPU indicano il rapporto tra il tempo che la CPU impiega per svolgere le operazioni richieste e il tempo totale trascorso. Se il rapporto si avvicina al 100%, significa che la CPU è ≪intasata≫.
Pannello (b): RAM e CPU utilizzate dal computer per caricare firefox e i contenuti di alcuni siti web (*web content*). Come potete vedere, la CPU è praticamente inutilizzata (il calcolatore non deve svolgere operazioni: il programma è già stato aperto) ma la conservazione dei dati occupa una vasta porzione di RAM (sono presenti parecchi dati in memoria).

1.2 Software

I *software* sono gli usuali programmi. Si differenziano dall'*hardware* perché essi possono essere scritti, modificati, trasferiti e cancellati molto facilmente. Un *software* (programma) può essere installato su un *hardware* (disco rigido).

1.2.1 Linguaggi di programmazione: il linguaggio macchina

Quando si parla di *software*, si studia prevalentemente il *linguaggio* che il computer utilizza per gestire i flussi di dati interni. Il **linguaggio fondamentale** (quello ≪compreso≫ direttamente dal computer) è il cosiddetto **linguaggio macchina**. Il **linguaggio macchina**, come qualsiasi linguaggio umano, è composto da **simboli** , da **parole** e da **frasi**.

I simboli del **linguaggio macchina** sono soltanto due: lo 0 e l'1. Per scrivere un comando di senso compiuto saranno quindi necessari moltissimi simboli: l'architettura x86 di molti computer prodotti negli anni 2000 utilizza fino a 120 simboli per ciascun comando.

Le **parole** sono sequenze di simboli, che possono avere vari significati. Le parole in informatica possono indicare **complementi oggetto** (numeri e caratteri), **predicati** (tutti coniugati all'imperativo: somma, sottrai, sposta, copia), e **complementi di luogo** (l'indirizzo di memoria).

Le **parole** possono comporre **frasi** (comandi di senso compiuto).Il numero di **simboli** necessari a formare una **frase** dipende dall'architettura della CPU. Se la CPU è a 8 bit, essa potrà leggere sequenze di 8 **simboli** e interpretarle come **parole**. Se la CPU è a 32 bit, essa potrà leggere sequenze di 32 **simboli** e interpretarle come **parole**.

Comandi operativi del processore ORB2018		
Linguaggio macchina	**assembly**	Linguaggio umano
1000 (prima parola)	SOM	Somma
1001 (prima parola)	SOT	Sottrai
1010 (prima parola)	CAR	Carica
1011 (prima parola)	SPO	Sposta
Indirizzi		
Linguaggio macchina	**assembly**	Linguaggio umano
0001 (seconda parola)	$1	Indirizzo della RAM numero 1
0111 (seconda parola)	$7	Indirizzo della RAM numero 7

Tabella 1.1: Comandi del processore ORB2018. Ovviamente, Bianca ha utilizzato l'italiano come lingua base per costruire il proprio **assembly**.

Ovviamente, per una persona è molto difficile comprendere il significato di un'istruzione in **linguaggio macchina** (**anche perché ciascun processore è caratterizzato da un linguaggio macchina tutto suo**). Immaginiamo ad esempio che Bianca Simoni inventi un processore tutto suo e che lo chiami ORB2018 (sigla che sta per «Orbetello 2018», se non fosse chiaro). Se il processore è a 4 bit, potrà interpretare **parole** lunghe 4 bit. Immaginiamo che il processore riesca a gestire **frasi** lunghe 5 parole. Un comando del processore ORB2018 apparirà come segue:

1000 0001 0010 0100

Tramite questo comando, il processore ORB2018 somma i numeri 2 e 4 e li salva all'indirizzo «1» della RAM. Non l'avevate intuito, vero? Ci credo!

1.2.2 Linguaggi assembly

Sarebbe comodo tradurre ciascuna **parola** contenuta nella **frase** riportata nel precedente paragrafo in un linguaggio più vicino a quello umano, che perlomeno sia composto da **parole** simili a quelle di lingua molto diffusa (per esempio l'inglese)

Questo è lo scopo dei linguaggi assembly: essi traducono *parola per parola* i linguaggi macchina delle CPU, utilizzando una notazione più familiare per un essere umano. Tuttavia, tali linguaggi mantengono le poco intuitive **sintassi** dei linguaggi macchina, e ciò li rende molto ostici.

Per comprendere il concetto, confrontiamo il comando visto nel paragrafo 1.2.1 utilizzando il **linguaggio macchina** e il **linguaggio assembly** creato da Bianca (per un riferimento, dai un'occhiata alla tabella 1.1):

1000 0001 0010 0100
SOM $1 2 4

I comandi **con lo stesso significato** sono stati evidenziati con lo stesso colore.

Cerchiamo intanto di comprendere la **sintassi** delle frasi del processore ORB2018, ovvero l'**ordine della parole**.

1. La prima **parola** è il **predicato**, ovvero l'azione che deve essere eseguita. In questo caso, l'azione che deve essere eseguita è una somma (SOM).
2. La seconda **parola** è il **complemento di luogo**, ovvero il luogo dove i dati devono essere salvati ($1, ovvero il primo indirizzo della RAM).
3. La terza e la quarta **parola** sono **complementi oggetto**, ovvero i dati che devono subire l'azione imposta dal **predicato**. Tali complementi oggetto sono i numeri 2 e 4.

Tenete conto del fatto che questa sintassi è arbitraria. Se Vittorio Restagno avesse costruito un suo processore (RESTx02), avrebbe potuto impostare una sintassi del tutto diversa.

Chiaramente, il **linguaggio assembly** continua a essere **molto ostico** per gli esseri umani, per vari motivi.

1. Prima di tutto, a noi non interessa assolutamente in quale indirizzo della RAM viene salvata la somma di due numeri. Vorremmo che ci pensasse il computer da solo a gestire i dati.
2. Inoltre, vorremmo effettuare operazioni un po' più complesse della somma o della sottrazione: vorremmo anche calcolare le potenze, scrivere automaticamente giganteschi vettori che contengano migliaia di numeri, salvare matrici che contengano tutti i pixel di un'immagine.
3. Infine, ci piacerebbe lavorare con un linguaggio **universale**, che non dipenda da quale CPU utilizza il computer.

Tutte queste esigenze possono essere soddisfatte da un linguaggio di programmazione di alto livello.

1.2.3 Linguaggi di alto livello

I **linguaggi di programmazione di alto livello** sono relativamente facili da comprendere e da utilizzare per comunicare i nostri desideri alla CPU. Infatti, per eseguire l'operazione che abbiamo visto nel pararagrafo 1.2.1 utilizzando un linguaggio di alto livello, sarà sufficiente scrivere:

$$V=2+4$$

La sintassi dei linguaggi di alto livello è molto vicina alla notazione matematica che utilizziamo usualmente. Sono quindi molto comodi per gli esseri umani.

Potremmo paragonare i linguaggi di alto livello all'italiano, i linguaggi assembly ai dialetti e i linguaggi macchina ai dialetti parlati da un signore anziano di 95 anni. Tramite il **linguaggio di alto livello** possiamo comunicare con tutti. Il **linguaggio assembly** ancora è comprensibile, ma ne esistono molte versioni, tante quante sono le CPU in circolazione. Anche il **linguaggio macchina** dipende dalla CPU, ma è praticamente incomprensibile per un essere umano.

Ovviamente, per portare completamente un testo dall'italiano al dialetto è necessario disporre di un **traduttore** o di un **interprete**.

Il traduttore cercherà di tradurre il testo integralmente e lo scriverà su un foglio, mentre l'interpete tradurrà **frase** per frase.

Anche in informatica funziona così: esistono dei programmi che traducono il nostro dal **linguaggio di alto livello** al **linguaggio macchina** (a volte utilizzando l'assembly come passo intermedio). Alcuni programmi (i **compilatori**) traducono il programma in **linguaggio macchina** e lo salvano sotto forma di **file eseguibile** (.exe su windows). Altri programmi (gli **interpreti**) traducono il programma in **linguaggio macchina** direttamente, senza salvare la traduzione in un file eseguibile.

Entrambe le operazioni di traduzione hanno vantaggi e svantaggi:

1. La compilazione è vantaggiosa perché produce un file già tradotto che il computer potrà eseguire rapidamente (è come se io vi chiedessi di leggere una poesia scritta in cinese già tradotta in italiano: sareste molto rapidi). È tuttavia svantaggiosa perché ci si può rendere conto se il programma funziona soltanto dopo averlo tradotto completamente. È come se io traducessi dal cinese un libro intero, vi chiedessi di leggerlo tutto e di trovare errori.

2. l'interpretazione è vantaggiosa perché permette di individuare rapidamente gli errori, traducendo un pezzetto di codice alla volta (questa operazione è detta *debugging*, ovvero eliminazione dei *bug*, ovvero degli errori). È tuttavia svantaggiosa perché non permette di creare un file in **linguaggio macchina** che il computer può leggere rapidamente

Un esempio di **linguaggio di alto livello** è MATLAB. Esso può essere sia interpretato che compilato. Voi, in laboratorio, finora lo avete soltanto *interpretato* e mai compilato.

Tutti i programmi che utilizzate tramite il vostro PC sono stati creati tramite un **linguaggio di alto livello** (come MATLAB, Python, Java, C++, Fortran).

1.3 Sintesi

Riassumiamo quello che abbiamo visto fino a questo punto:

1. l'informatica studia le metodologie di gestione delle informazioni.

2. La gestione delle informazioni è divisa in tre parti: **lettura**, **elaborazione** e **immagazzinamento**.

3. Il computer è composto da varie parti di *hardware*, ciascuna delle quali può essere utilizzata per la **lettura**, per l'**elaborazione** o per l'**immagazzinamento** .

4. La **lettura** è l'acqusizione degli *input*, mentre l'**immagazzinamento** è il salvataggio degli *output*. L'**elaborazione** è l'utilizzo degli *input* per produrre determinati *output*.

5. l'*hardware* del computer può ospitare dei *software*, cioè dei programmi, i quali dicono alla CPU cosa deve fare.

6. I *software* sono scritti in **linguaggio macchina**.

7. l'uomo non può scrivere in **linguaggio macchina**, perché ne esistono molte varietà e perché è poco intuitivo e difficile da gestire. È necesssario scrivere dei linguaggi intermedi (**assembly**) e dei linguaggi più intutivi e facilmente gestibili (**linguaggi di alto livello**) per poter lavorare direttamente e agevolmente con il computer.

8. I testi scritti tramite **linguaggi di alto livello** devono essere tradotti in **linguaggio macchina**, altrimenti il calcolatore non ne può comprendere il senso.

9. La traduzione può essere effettuata da due tipi di programmi: i **compilatori**, che producono degli eseguibili salvandoli in memoria, e gli **interpreti**, che traducono i testi comando per comando.

Sezione 2

MATLAB

MATLAB è un linguaggio di programmazione. Esiste una comoda interfaccia utente chiamata anch'essa MATLAB, che voi avete utilizzato in laboratorio. Tale interfaccia può essere considerata un «contenitore» di programmi, come l'**editor** di testo che vi permette di scrivere i vostri *codici sorgente* in linguaggio MATLAB.

MATLAB è un linguaggio di programmazione particolarmente adatto ai principianti. **Si può dire che un linguaggio di programmazione è tanto più adatto ai principianti quante più sono le librerie disponibili in tale linguaggio**. Le **librerie** e sono dei piccoli programmi inclusi nel «pacchetto» del linguaggio di programmazione. È ovvio che disporre di tante **librerie** e facilita la vita: immaginate di dover scrivere un programma che calcoli la potenza di un numero disponendo soltanto della funzione «somma». Il programma risulterebbe piuttosto complicato! È chiaro che disporre di una funzione «potenza» già pronta rende le cose più rapide. MATLAB dispone di innumerevoli funzioni precalcolate di elevatissima complessità, che vedremo con il passare dei mesi.

2.1 Variabili semplici e formati

MATLAB può gestire una lunga serie di variabili semplici. Ne abbiamo già viste parecchie, tra le quali si annoverano le seguenti:

1. uint8, ovvero numero **intero senza segno** (da 0 a 255)
2. int8, ovvero numero **intero con segno** (da -128 a 127)
3. single, ovvero numero **a virgola mobile** a 32 bit.
4. double, ovvero numero **a virgola mobile** a 64 bit.
5. char, ovvero caratteri a 11 bit.

Abbiamo visto più volte in classe che ogni numero è caratterizzato da uno specifico **formato**.

Alcuni di questi formati sono abbastanza «monolitici», ovvero non sono caratterizzati da un elevato grado di arbitrarietà. I formati «interi senza segno», per esempio, rappresentano praticamente sempre numeri che vanno da 0 a $2^N - 1$, dove N è il numero di bit della rappresentazione.

2.1.1 Interi con segno

Altri formati invece sono piuttosto «volatili», perché sono caratterizzati da un elevato grado di arbitrarietà. Ad esempio, i numeri **interi con segno** possono essere rappresentati tramite numerosi formati. Assumendo di lavorare con numeri a 8 bit, si ottiene la seguente scaletta:

1. Il formato «complemento a due». Per ottenere il numero corrispondente in base 10, si seguono i passi sottostanti:
   (a) Se la prima cifra è 0, si converte direttamente come si farebbe con un **intero senza segno**.
   (b) Se la prima cifra è 1, si sottrae 256 al risultato della conversione.

   In formule:
   $$N_{10} = (b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8)_2 - b_1 \cdot 256$$
   dove $b_i,\ i \in \mathbb{N}$ sono i *bit* che compongono i numeri. Questi numeri vanno da -128 a 127.
2. Il formato «interno». Per ottenere il numero corrispondente in base 10 è necessario convertire il numero N come se fosse un **intero senza segno** e poi sottrarre 127. In formule:
   $$N_{10} = (b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8)_2 - 127$$
   Questi numeri vanno da -127 a 128.

3. Il formato «esponenziale». Per ottenere il numero corrispondente in base 10 è necessario convertire le ultime 7 cifre dalla base 2 alla base 10 e poi apporre un segno meno davanti al risultato se la prima cifra è 1, e un segno più se la prima cifra è 0. In formule:

$$N_{10} = (-1)^{b_1}(b_2b_3b_4b_5b_6b_7b_8)_2$$

Questi numeri vanno da -127 a 127. **Il numero 0 è rappresentato da due sequenze (0000 000 e 1000 000)**.

2.1.2 Numeri a virgola mobile

I numeri a virgola mobile, in linea di principio, possono essere rappresentati tramite **un'infinità di formati**. Lo scopo dei numeri a **virgola mobile** è rappresentare un ampio intervallo di valori tramite una notazione esponenziale, e soprattutto **includere i numeri razionali**.

Esiste uno standard (il binary32) che impone l'utilizzo di numeri a **virgola mobile** composti da 32 bit, ma esso è forse troppo complesso da trattare per voi (troppi bit)!

Preferisco quindi esporvi il **metodo generale** per scrivere numeri a virgola mobile. Essi sono composti dal prodotto di due fattori: la **mantissa** (M) e un **fattore esponenziale** (E):

$$\mathrm{N}_{10} = M \cdot E = (P, D) \cdot E = (P, D) \cdot 2^e \tag{2.1}$$

La mantissa è composta da una parte prima della virgola (P) e una parte dopo la virgola (P). Il fattore esponenziale «E» è una potenza di 2.

La parte prima della virgola (P) è generalmente composta da un solo bit, che indica il segno. La parte dopo la virgola (D) è una serie di cifre decimali. L'esponente «e» è un numero intero con segno.

Un esempio di formato a **virgola mobile** (che definiremo **virgola mobile orbetellana**) è il seguente:

$$\mathrm{N}_{10} = (-1)^{b_1} \cdot (1, b_2b_3b_4b_5b_6b_7)_2 \cdot 2^{(b_8b_9b_{10}b_{11}b_{12})_2 - 16_{10}} \tag{2.2}$$

Questo formato è composto da **12 bit** In questo caso abbiamo:

1. $P = (-1)^{b_1}$
2. $D = (1, b_2b_3b_4b_5b_6b_7)_2$
3. $e = (b_8b_9b_{10}b_{11}b_{12})_2 - 16_{10}$

Per comprendere il significato del formato, proviamo a esprimere il numero 1001 0011 1111_{vmo} in base 10:

$$\begin{aligned}
\mathrm{N}_{10} &= (-1)^1 \cdot (1, 001001)_2 \cdot 2^{11111_2 - 16_{10}} && (2.3)\\
&= -1 \cdot (1, 001001)_2 \cdot 2^{(31-16)_{10}} && (2.4)\\
&= -1 \cdot (1, 001001)_2 \cdot 2^{15_{10}} && (2.5)\\
&= -1 \cdot (1001001)_2 \cdot 2^{9_{10}} && (2.6)\\
&= -1 \cdot (1 + 8 + 64)_{10} \cdot 512_{10} && (2.7)\\
&= -1 \cdot 73_{10} \cdot 512_{10} && (2.8)\\
&= -37376_{10} && (2.9)
\end{aligned}$$

2.2 Vettori

I vettori sono **contenitori di variabili semplici**.

I vettori sono molto utili, perché permettono di salvare una grossa quantità di dati affini in una sola variabile.

Su MATLAB, è possibile definire i vettori in vari modi:

Definizione manuale L'utente decide di inserire una serie di valori arbitrari in un vettore. Per definire manualmente un vettore è necessario utilizzare la seguente sintassi:

V=[v1 v2 v3 v4 v5...]

Esempio:

V=[3 7 2 0 1 4]

dove **v1, v2, v3, v4** e **v5** sono variabili semplici.

Campionamento costante L'utente decide di inserire una serie di valori separati da un intervallo costante. La sintassi è la seguente:

V=vi:int:vf

dove **vi,int** e vf sono variabili semplici che indicano il valore iniziale, l'intervallo di campionamento e il valore finale.
Esempio:

V=-7:0.5:-3

Otterremo:

V=[-7 -6.5 6 -5.5 -5 -4.5 -4 -3.5 -3]

Vettore vuoto L'utente decide di creare un vettore vuoto di lunghezza arbitraria. La sintassi è la seguente:

V=zeros(L,1)

dove **L** è una variabile che indica la lunghezza del vettore. Esempio:

V=zeros(13,1)

Otterremo:

V=[0 0 0 0 0 0 0 0 0 0 0 0 0]

Vettore pieno di uno L'utente decide di creare un vettore pieno di uno di lunghezza arbitraria. La sintassi è la seguente:

V=zeros(L,1)

dove **L** è una variabile che indica la lunghezza del vettore. Esempio:

V=ones(5,1)

Otterremo:

V=[1 1 1 1 1]

2.2.1 Operazioni con i vettori

Le operazioni con i vettori sono diverse dalle operazioni con le variabili semplici. Prendiamo ad esempio la moltiplicazione: essa può essere definita in molti modi. **Il prodotto tra due vettori può essere scalare, vettoriale, o semplicemente prodotto tra elementi che occupano posizioni corrispondenti (prodotto elemento per elemento)**.

Queste ambiguità richiedono una **definizione rigorosa degli operatori**.

1. Supponiamo di voler elevare tutti gli elementi di un vettore al quadrato. In questo caso, sarà necessario moltiplicare ciascun elemento del vettore per gli elementi corrispondenti di un vettore identico. Su MATLAB, tale operazione è definita tramite la seguente notazione:

   V=U.*U

   Esempio:

   V=[3 2 3].*[3 2 3]
   V=[9 4 9]

   In pratica, il **prodotto elemento per elemento** si ottiene scrivendo un punto e un asterisco.

2. Supponiamo ora di voler calcolare il prodotto scalare tra due vettori (questo è necessario qualora si voglia determinare quantità come il lavoro, che è definito come il prodotto scalare tra forza e spostamento: $L = F \cdot s$). Su MATLAB, tale operazione è definita tramite la seguente notazione:

   V=U*W

   Esempio:

   V=[10 8 7]*[3 2 5]'
   V=$10 \cdot 3 + 8 \cdot 2 + 7 \cdot 5$**=81**

   Attenzione: qualora si desideri effetture un prodotto scalare, il primo vettore deve essere orizzontale (*vettore riga*), e il secondo deve essere verticale (*vettore colonna*). **Per ottenere un vettore riga a partire da un vettore colonna e viceversa, sarà sufficiente apporre un apice (') dopo la parentesi quadra che chiude il vettore**.

Sezione 3

Esercizi di preparazione al compito del 21 dicembre 2018

Svolgete i seguenti esercizi.

1. Esprimi il numero N_1 in base 10 ($N_1 = 0010\ 0011\ 1111$) sapendo che N_1 è un intero senza segno.
2. Esprimi il numero N_2 in base 10 ($N_2 = 1000\ 0000\ 1100$) sapendo che N_2 è un intero con segno che segue il formato sottostante:

$$N_2 = (-1)^{b_1}(b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12})_2$$

3. Supponi di aver definito sul tuo computer una variabile N_3, un intero con segno a 8 bit con il seguente formato:

$$M = (-1)^{b_1}(b_2 b_3 b_4 b_5 b_6 b_7 b_8)_2 \tag{3.1}$$

   Nello spazio di memoria dedicato a tale variabile (che non può occupare più di 8 bit) desideri caricare la somma dei numeri interi senza segno M_1 e M_2, che hanno lo stesso formato di M. Tu sai che $M_1 = 1000\ 1010$ e che $M_2 = 1111\ 1100$.

   (a) Come potresti definire la somma di questi due numeri utilizzando una notazione simile a quella dell'equazione 3.1?

   (b) A tuo parere, quale sarà il valore che apparirà nell'indirizzo di memoria della variabile M?

4. I numeri in **formato esadecimale** sono espressi tramite un formato posizionale del tutto affine a quello binario e decimale. L'unica differenza è che il formato esadecimale dispone di 16 simboli:

   0 1 2 3 4 5 6 7 8 9 A B C D E F

   (a) A tuo parere, quanti byte di informazioni sono necessari per immagazzinare una cifra esadecimale, sapendo che un byte è uguale a 8 bit?

   (b) Utilizzando 8 cifre esadecimali, quante cifre binarie posso rappresentare?

   (c) Esprimi il numero AB_{16} tramite il codice binario.

5. Determina il valore del numero $N_4 = 1111\ 1111\ 1111$, sapendo che tale numero è stato scritto tramite il formato a virgola mobile orbetellano.