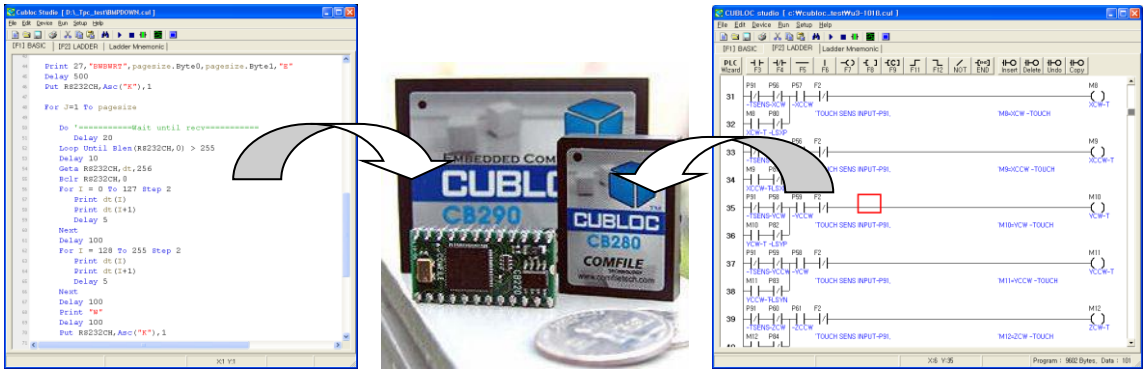
Istituto **P**rofessionale per L'**i**ndustria L'**A**rtigianato

63039 SAN BENEDETTO DEL TRONTO (AP)

Classe 5A T.I.E.N. Anno Scolastico 2012-2013

MICROCONTROLLORI CUBLOC

Software



Sistemi, automazione e organizzazione della produzione

1.1 Introduzione

I microcontrollori sostituiscono in misura sempre maggiore i tradizionali circuiti di regolazione, misura, e controllo. Essi sono caratterizzati da una uguale struttura hardware: CPU, ROM, RAM, PORTE.

Tramite programmazione, i micro si trasformano in integrati dedicati o specializzati alla gestione di una sola applicazione, ne consegue che l'attività maggiore di progettazione non è hardware, bensì software.

Infatti, il costo delle ore di lavoro necessarie al programmatore per "specializzare" il micro, ovvero per scrivere il programma, risulta decisamente superiore al costo del micro stesso, ecco quindi l'esigenza di razionalizzare il lavoro seguendo un criterio di programmazione ben preciso.

1. **Analisi del problema e fattibilità**: identificazione di una possibile soluzione, verifichiamo se le risorse presenti nel micro e il set d'istruzioni consentono di soddisfare tutte le richieste dell'applicazione in esame.
2. **Schema elettrico**: stabiliamo i collegamenti tra il micro e il mondo esterno, associamo ad ogni porta la relativa funzione e il modo di funzionamento (ad esempio: ingresso, uscita, ingresso analogico, ecc.)
3. **Tabella di verità:** associamo ad ogni porta il relativo compito, ad esempio: P1 è un ingresso con resistore pull-up collegato ad un interruttore e risulta attivo quando assume il valore di "0" logico, ecc.
4. **Flow_Chart**: realizziamo lo schema a blocchi (flow-chart) del programma, pianificando la sequenza delle istruzioni in funzione di eventi interni o esterni, si trova l'**algoritmo**
5. **Programma**: tramite un Computer con l'ausilio del software Cubloc Studio si traduce il Flow Chart in programma (Basic oppure Ladder), rispettando il set di istruzioni del micro, si crea il file sorgente.
6. **Compilazione e memorizzazione**: tramite lo Study Board (Starter Kit) e il software Cubloc Studio il file sorgente viene compilato e trasferito nella memoria programma del microcontrollore. Lo Starter Kit permette tramite la porta USB il collegamento tra microcontrollore e Personal Computer.
7. **Collaudo**: si effettua il collaudo finale del progetto, si inserisce il micro programmato nel circuito progettato, comprensivo di tutte le sue interfacce (clock, reset, alimentazione, I/O) e si verifica se il funzionamento corrisponde alle specifiche richieste dal progetto.

1.2 Algoritmo – Flow-Chart

Dato un problema, un **algoritmo** è una procedura, cioè una sequenza di passi, che può essere eseguita automaticamente da una macchina in modo da risolvere il problema dato.

Un problema risolvibile mediante un algoritmo si dice *computabile*.

L'insieme dei valori permessi ai dati in ingresso si definisce **dominio** dell'algoritmo, mentre l'insieme dei valori che possono assumere i dati in uscita rappresenta il **codominio**.

Un **programma** prende origine comunque da un **algoritmo** che viene descritto in modo sintetico utilizzando un tipo di rappresentazione, la maniera più comune di rappresentazione è quella dei diagrammi di flusso o **flow-chart**.

Il flow-chart rappresenta l'algoritmo graficamente mediante una lista di istruzioni inserite in appositi simboli grafici, ogni simbolo viene collegato al successivo con delle linee orientate che danno luogo così alla sequenza delle operazioni da effettuare.

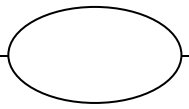
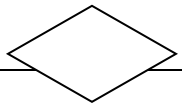
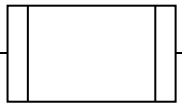
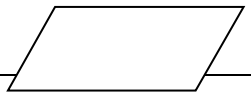
Il flow-chart o diagramma a blocchi da un visione immediata dell'intero algoritmo mettendo in evidenza la sequenza di esecuzione delle varie operazioni.

Un sistema programmabile (computer, microcontrollore, ecc) non ha una propria capacità di "elaborazione creativa", cioè non è in grado di eseguire alcuna operazione se non viene opportunamente istruito. Spetta quindi a noi tradurre il problema in termini formali, individuare dati e incognite (gli elementi non noti, da determinare), schematizzare tutti i passaggi, prevedere tutti i possibili casi che si possono presentare e indicare opportunamente la via da seguire, in modo che la macchina possa arrivare alla soluzione.
L'individuazione di una sequenza ordinata di istruzioni che porta alla risoluzione di un problema viene definita **algoritmo**.
Utilizziamo gli algoritmi, per esempio, quando prepariamo una pietanza in cucina, per montare un giocattolo, per sommare due numeri o per effettuare l'iscrizione a scuola.
Le operazioni definite devono poi essere tradotte in opportuni linguaggi, in modo che le istruzioni possano essere "comprese" dal Sistema programmabile (**esecutore**).
L'informatica si occupa della risoluzione di problemi mediante algoritmi.

Le istruzioni di un algoritmo devono essere:

- interpretabili in un unico modo, non possono essere ambigue;
- elementari; quelle più complesse devono essere suddivise in istruzioni più semplici,in modo che possano essere "capite" più facilmente dal sistema programmabile;
- in numero finito; se così non fosse, come potrebbe un sistema programmabile arrivare a una conclusione? Lavorerebbe all'infinito senza poter restituire alcun risultato.

In tabella sono rappresentati i simboli principali per la stesura di un flow-chart.

Simbolo	Descrizione
	Inizio di un programma o di una subroutine, si scrive all'interno il nome del programma o della subroutine, viene anche utilizzato per indicare fine programma o fine subroutine
	Elaborazione dati, esempio calcolo, oppure manipolazione dati, operazione incondizionata
	Blocco decisione, esso contiene una domanda, in funzione della risposta viene deviato il corso del flow-chart, tale blocco è caratterizzato da una una linea d'ingresso e da due diverse linee di uscita (Vero, Falso).
	Chiamata subroutine
	Operazione Input /Output
	Flusso logico

Per realizzare un flow-chart scriviamo una o più azioni/comandi all'interno di figure geometriche (ellissi, rettangoli, rombi, ecc) e colleghiamo le figure tra loro con delle linee,

aggiungiamo ad ogni linea di congiunzione una freccia, essa indica la sequenza degli eventi, ovvero la sequenza utilizzata dal micro per processare le istruzioni.

1.2.1 Algoritmo non numerico

Prova a pensare alle operazioni che fai quando ti prepari una camomilla:

- scaldi l'acqua,
- sistemi un filtro in una tazza,
- versi nella tazza l'acqua calda,
- lasci in infusione per qualche minuto,
- aggiungi zucchero.

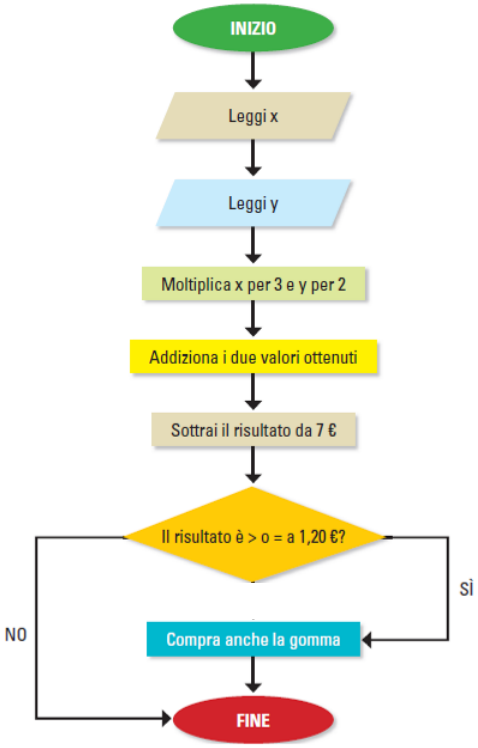
La sequenza di passi appena descritta costituisce un algoritmo, in figura il relativo diagramma di flusso.

1.2.2 Algoritmo numerico

Creare l'algoritmo che rappresenta il seguente problema:
Devo acquistare tre penne che hanno un costo x e due matite di costo y.
Se ho a disposizione 7 € mi rimangono i soldi per comprarmi anche una gomma da 1,20 €?

- Possiamo definire il seguente algoritmo:
- moltiplica per tre il costo di una penna;
- moltiplica per due il costo di una matita;
- addiziona i due valori ottenuti (totale della spesa per le penne e per le matite);
- confronta la differenza tra quanto hai a disposizione (7 €) e la somma dei due valori: se è maggiore (>) o uguale (=) al costo della gomma (1,20 €) puoi comprarla, altrimenti no.

Quello appena descritto è l'algoritmo espresso in termini informali.
Formalizziamo i passaggi con il seguente diagramma di flusso.



Algoritmo numerico

Algoritmo non numerico

1.3 Gli algoritmi ed i programmi

Un sistema programmabile (Computer, microcontrollore, ecc) è una macchina in grado di elaborare dei dati in funzione di una determinata logica definita da qualcuno. La descrizione della logica in base alla quale il sistema deve elaborare i dati è detta **algoritmo** e consiste in un numero finito di passi da seguire per arrivare alla soluzione. Un algoritmo può essere espresso in varie forme, ma quelle più utilizzate sono essenzialmente due:

- linguaggio naturale
- diagramma a blocchi (Flow Chart)

Esistono essenzialmente due tecniche per la stesura di un algoritmo che hanno caratteristiche opposte

- **top-down** è l'approccio più classico. È il metodo secondo il quale chi scrive l'algoritmo deve avere da subito un'idea della struttura dell'algoritmo stesso: la stesura dell'algoritmo inizia dal suo inizio e procede sequenzialmente fino ad arrivare alla fine;
- **bottom-up** è l'approccio nato per la stesura di algoritmi relativi a problemi più complessi. Si basa sul motto latino "divide et impera", ovvero il problema viene spezzato in sottoproblemi più facili da risolvere e si inizia a stendere l'algoritmo per questi. Quindi si combinano insieme le varie parti fino ad ottenere l'algoritmo che risolve il problema nel suo complesso.

Nella realtà per la stesura di un algoritmo si fa generalmente uso di entrambe le due tecniche: per lo sviluppo delle parti per le quali è necessario avere una visione d'insieme del problema si utilizza la tecnica top-down, mentre per sviluppare le parti per le quali si sente più l'esigenza di concentrarsi sul dettaglio, si utilizza la tecnica bottom-up.

Un algoritmo, sia esso espresso in linguaggio naturale che sottoforma di diagramma a blocchi, descrive i passi necessari per la soluzione di un problema, ma non è comprensibile da un computer. A tal fine esso deve essere trasformato in un **programma**. Un programma infatti non è altro che l'implementazione di un algoritmo in un linguaggio comprensibile all'elaboratore, cioè una sequenza finita ed ordinata di istruzioni eseguibili in un tempo finito.

La strutturazione dell'algoritmo in più parti porta alla stesura di un programma suddiviso in diversi blocchi.

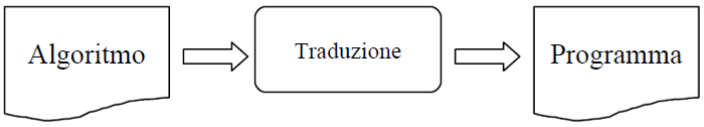
1. programma principale detto "***main program***"
2. uno o più programmi secondari denominati "***subroutine***".

Per meglio localizzare eventuali errori di stesura del programma è consigliabile sviluppare e provare le subroutine singolarmente, e solo quando queste ultime funzioneranno correttamente sarà possibile unirle al main program per testare il programma completo.

È buona norma, inoltre, realizzare le subroutine e catalogarle formando una biblioteca di programmi secondari.

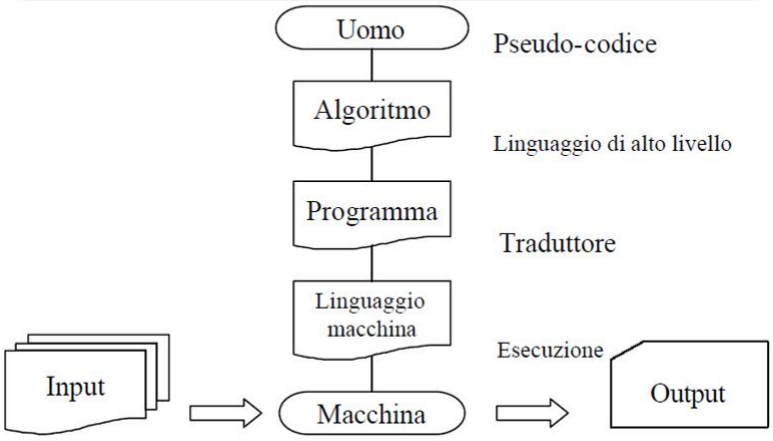
1.3.1 codifica dell'algoritmo

Affinché una macchina riesca a comprendere ed eseguire i passi specificati da un algoritmo quest'ultimo deve essere prima codificato in un opportuno programma scritto in un linguaggio di alto livello.



La CPU è in grado di comprendere solo un particolare insieme di istruzioni: il **linguaggio macchina** (o **codice macchina**) composto da particolari sequenze di valori binari. Scrivere un programma in linguaggio macchina non è né pratico né agevole. Ecco quindi che sono nati i linguaggi di programmazione, dei linguaggi intermedi, molto vicini al linguaggio naturale (rispetto al linguaggio macchina), che rendono più facile la vita al programmatore. Una volta scritto un programma con il linguaggio di programmazione desiderato esistono degli appositi programmi che fanno la "traduzione" del programma in linguaggio macchina, in maniera da renderlo comprensibile alla CPU e, quindi, eseguibile.

L'esecuzione automatica



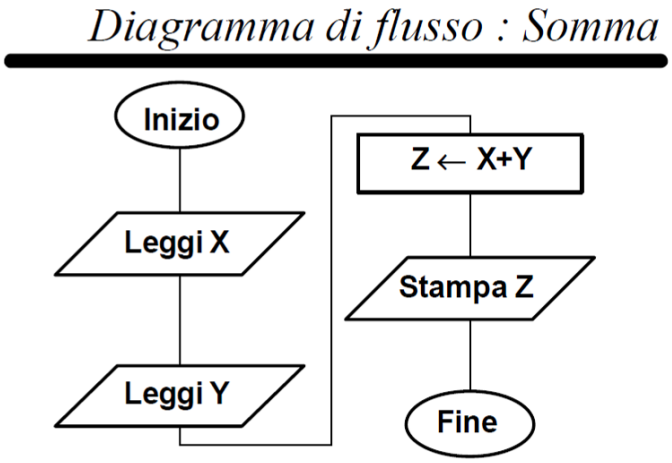
Esistono essenzialmente due metodi per effettuare la traduzione di un programma in linguaggio macchina: l'*interpretazione* e la *compilazione.*

Istruzione di assegnamento

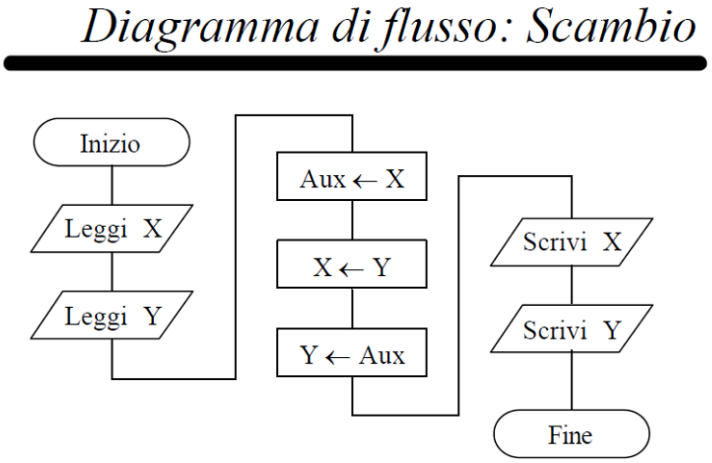
- Una *variabile* numerica è un'entità caratterizzata :
  - da un nome, e
  - da un valore (o contenuto)
    - che può cambiare nel tempo …
- Un'*espressione* è una combinazione di operatori aritmetici, costanti e variabili che può essere calcolata generando un singolo valore numerico.
  - Es.: X, X + 1, X + (Y * 3).
- Istruzione di *assegnamento* : " ← "
  - Variabile ← Espressione;
    - Es.: Z ← 3; Z ← X + 3; X ← X + 1;

1.4 Esempi

1) Descrivere, mediante diagramma di flusso, **un** algoritmo che calcoli la *somma* di due numeri letti in input.



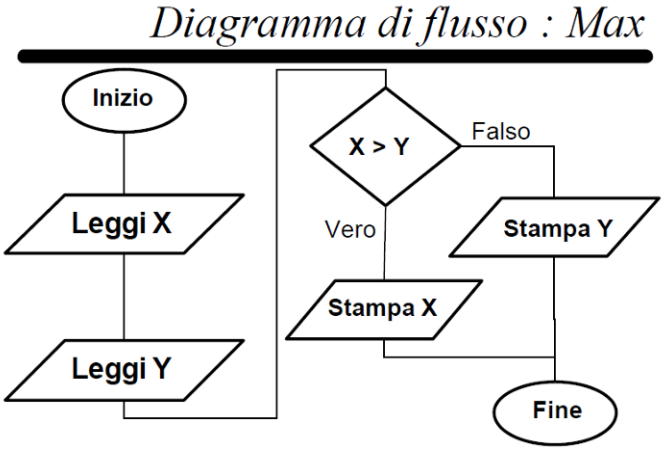
2) Descrivere, mediante diagramma di flusso, un algoritmo che scambi i valori di due variabili lette in input.



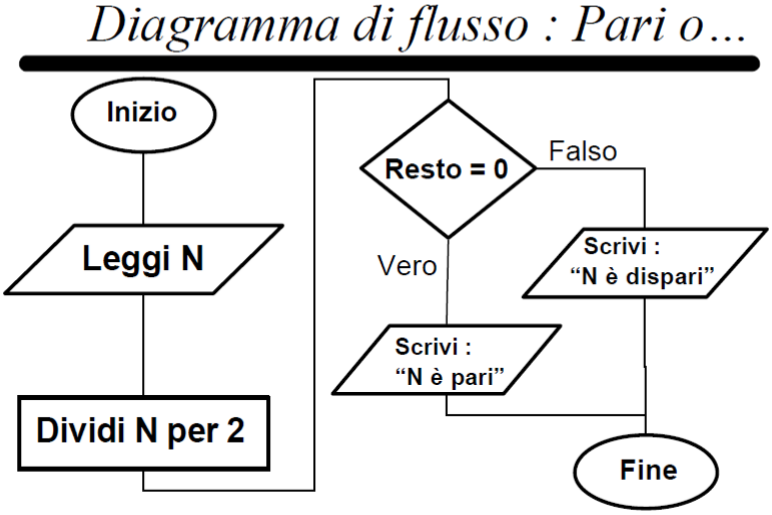
Variazioni nel flusso di esecuzione

- Ci sono dei momenti in cui il flusso di esecuzione può scegliere tra diverse direzioni;
- In genere, questi *salti* sono subordinati al verificarsi di una *condizione* (che può risultare *vera* o *falsa*);
- Si parla di *istruzioni condizionali*.

3) Descrivere, mediante diagramma di flusso, un algoritmo che determini il *massimo* tra due numeri letti in input.



4) Descrivere, mediante diagramma di flusso, **un** algoritmo che determini se un numero letto in input è *pari* o *dispari*.



1.5 Programmazione Basic dei moduli CUBLOC

L'acronimo BASIC sta per "Beginner's All purpose Symbolic Instruction Code", cioè "codice di istruzioni simboliche di uso generale per principianti".
La prima versione fu progettata nel 1962 da Thomas Kurtz e John Kemeny allo scopo di creare un linguaggio semplice da imparare anche per i dilettanti dell' informatica e per questo si differenziò dalla maggior parte dei linguaggi suoi coetanei; nella realizzazione del BASIC si è privilegiata la semplicità d'uso rispetto alla potenza espressiva. Una delle versioni più famose è il Microsoft BASIC, sviluppato da Bill Gates e Monte Davidoff, poi evolutasi nel GW-BASIC preparato per il primo MS-DOS. Con gli anni il BASIC hi subito notevoli evoluzioni e cambiamenti, diventando un linguaggio strutturato con potenzialità molto simili a quelle di altri linguaggi evoluti come il Pascal ed il C le numerose versioni attuali del BASIC, come ad esempio il Visual Basic, affiancano a quelli originari nuovi concetti quali la gestione degli oggetti, la ricorsione, 1a programmazione strutturata, ecc. Anche la versione e BASIC supportata dai moduli CUBLOC presenta delle peculiarità che lo differenziano dallo standard, le più importanti delle quali sono riassunte nella Tabella 1

Tabella **1 - Basic standard e Basic Cubloc a confronto**

	BASIC STANDARD	BASIC CUBLOC
Struttura delle istruzioni	Precedute da una etichetta progressiva (numero di riga).	I numeri di riga non vengono più utilizzati; è possibile definire le label.
Modo di funzionamento	Interpretato.	Compilato.
Operatori aritmetici	Elevamento a potenza (^), cambio di segno (-), moltiplicazione (*), divisione (/), addizione (+), sottrazione (-), modulo (MOD).	Elevamento a potenza (^), cambio di segno (-), moltiplicazione (*), divisione (/), addizione (+), sottrazione (-), modulo (MOD). Esistono anche funzioni di libreria per l'incremento e decremento.
Operatori relazionali	Uguaglianza (=), disuguaglianza (< >), minore (<), maggiore (>), minore o uguale (< =), maggiore o uguale (> =).	Uguaglianza (=), disuguaglianza (< >), minore (<), maggiore (>), minore o uguale (< =), maggiore o uguale (> =).
Operatori logici	NOT, AND, OR, XOR.	NOT, AND, OR, XOR.
Operatori sui bit	non previsti	NOT, AND, OR, XOR, shift a sinistra (< <), shift a destra (> >).
Costanti	È possibile utilizzare soltanto costanti numeriche.	Oltre all'utilizzo delle costanti numeriche è possibile definire costanti mediante l'istruzione CONST.
Variabili	Non devono essere dichiarate; possono essere di tipo intero, singola e doppia precisione, stringa. Il tipo è specificato da un carattere speciale che precede il nome della variabile (%, !, #, $).	Devono essere dichiarate; sono supportati il tipo BYTE, INTEGER, LONG, SINGLE, STRING.
Variabili strutturate	Array, stringhe (specificate dal simbolo $).	Array, stringhe. Sono inoltre presenti operatori per l'accesso diretto a bit, nibble, byte, word.
Istruzioni di controllo del programma	FOR .. NEXT, WHILE .. WEND, GOTO, ON GOTO.	For .. Next, Do .. Loop, Goto (utilizzo sconsigliato),
Istruzioni condizionali	IF .. THEN .. ELSE.	If .. Then .. Else .. End If, Select .. Case.
Istruzioni per la gestione dei sottoprogrammi	GOSUB, RETURN.	Gosub, Return, Sub, Function.
Visibilità delle variabili	Globale.	Globale o locale, in base alla posizione di dichiarazione.
Passaggio dei parametri	non previsti	Per valore.
Funzioni	Sono di tipo numerico (ABS, FIX, INT, SQR, RND, SIN, COS, TAN, ATN, EXP, LOG, ecc..) e stringa (ASC, CHR$, LEFT$, RIGHT$, MID$, LEN, STR$, ecc..).	Oltre alle funzioni standard, esistono particolari librerie legate all'hardware del modulo.
Preprocessore	non previsti	Sono supportate le direttive #include, #define, #if .. #endif.

1.5.1 Linguaggi di programmazione

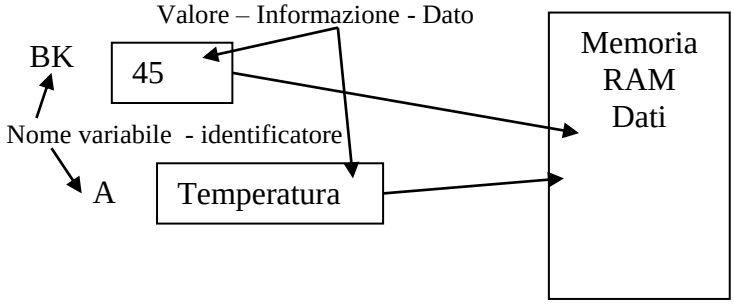
I linguaggi di programmazione sono un insieme di parole e di regole, definite in modo formale, che consentono la programmazione di un elaboratore elettronico in modo che possa eseguire problemi o, più precisamente, algoritmi. Si definisce algoritmo una sequenza di istruzioni che risolve un problema in un numero finito di passi. È possibile classificare i linguaggi di programmazione in funzione del loro livello di astrazione (cioè di quanto siano rivolti alla macchina piuttosto che all'uomo), in linguaggi di basso, medio ed alto livello. Quello di livello più basso in assoluto prende il nome di linguaggio macchina ed è interpretato direttamente dall'elaboratore; è costituito da sequenze di numeri binari corrispondenti ai set di istruzioni del microprocessore. Nel linguaggio macchina si fa riferimento direttamente all'architettura dei calcolatore e pertanto si raggiunge il massimo livello di efficienza, sia in termini di velocità di esecuzione che di occupazione di memoria, ma i programmi sono quasi del tutto incomprensibili per il programmatore e non sono portabili, cioè non possono essere eseguiti su di una macchina differente da quella per cui sono stati scritti. I linguaggi di alto livello, fra cui il BASIC, perdono sì di efficienza, ma agevolano il lavoro del programmatore, che può implementare più facilmente gli algoritmi. Infatti le parole chiave di un linguaggio ad alto livello sono espresse generalmente da termini in lingua inglese che esprimono l'azione svolta. Un linguaggio di medio livello è il cosiddetto codice mnemonico, nel quale un'istruzione consta in una o più istruzioni in linguaggio macchina. I linguaggi di medio e alto livello necessitano di un interprete specifico per la CPU.

1.5.2 Le variabili

Nei linguaggi di programmazione si definisce variabile una locazione di memoria dati cui è associato un nome e che è utilizzata per memorizzare un valore che può essere modificato da un programma. I nomi mediante i quali è possibile fare riferimento alle variabili prendono il nome di identificatori.

In BASIC, un identificatore è costituito da uno o più caratteri il primo dei quali deve essere una lettera e può essere seguito da un qualsiasi carattere alfanumerico.

Le costanti sono invece dei valori che non possono essere alterati durante l'esecuzione del programma.



1.5.3 Tipi di dato, identificatori, variabili e costanti

Per la memorizzazione delle variabili, il BASIC CUBLOC definisce cinque tipi di base: Byte, Integer, Long, Single e String

- **Byte**: viene utilizzato per memorizzare valori interi positivi di otto bit (0 .. 255);
- **Integer**: per valori interi positivi di dimensione pari a due byte (0 .. 65535);
- **Long**: Long serve per valori interi positivi o negativi di quattro byte (-2147483648 .. 2147483647);
- **Single**: viene impiegato per valori reali a singola precisione (-3.402823 10^{38} ... 3.402823 10^{38}) utilizza 32 bit per la memorizzazione del numero).
- **String** si usa per sequenze di caratteri di lunghezza massima di 127 byte.

Le variabili devono essere dichiarate prima del loro utilizzo nel programma.

Per la dichiarazione delle variabili si usa il comando DIM oppure Var

Comando DIM (sintassi ed esempi)

DIM *nome variabile* As *tipo variabile*

dove *tipo variabile* è uno dei tipi precedentemente menzionati, mentre la *nome variabile* è costituita da uno specifico identificatore.

Esempio

Dim A As byte *'dichiara la variabile di nome A di tipo byte*
Dim B As integer *'dichiara la variabile di nome B di tipo Integre*

```
Dim A As Byte                    'Declare A as BYTE.
Dim B As Integer, C As Byte            'Comma may NOT be used.
Dim ST1 As String * 12           'Set String size for String.
Dim ST2 As String                'Set as 64 bytes (default).
Dim AR(10) As Byte               'Declare as Byte Array.
Dim AK(10,20) As Integer         'Declare as 2D Array
Dim ST(10) As String*10          'Declare a String Array
```

Comando VAR (sintassi ed esempi)

nome variabile Var *tipo*

```
A         Var    Byte          ' Declare A as BYTE.
ST1       Var    String * 12   ' Declare ST1 as String of 12 bytes.
AR        Var    Byte(10)            ' Declare AR as Byte Array of 10.
AK        Var    Integer(10,20)      ' Declare AK as 2-D Integer Array
ST        Var    String *12 (10)     ' Declare String Array
```

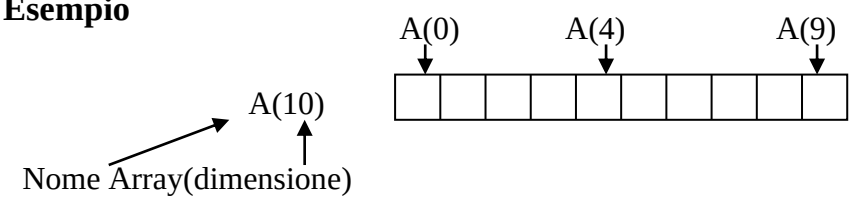
La dichiarazione della stringa alloca, per impostazione predefinita (default) 64 caratteri, è comunque possibile specificare mediante il simbolo * il numero dei caratteri da cui è composta.
Esempio ST1 Var String*12 ‘ dichiara una variabile stringa in grado di memorizzare massimo 12 caratteri (un carattere equivale ad un byte)

Il simbolo ‘ indica che tutto ciò che segue sulla riga è un commento e viene ignorato dal compilatore.

Array

Un Array è costituito da un insieme di caselle (Vettore, matrice, tabella), le singole caselle sono dette celle dell'array. Ciascuna delle celle si comporta come una variabile tradizionale; tutte le celle sono variabili di uno stesso tipo preesistente, detto tipo base dell'array.

Esempio

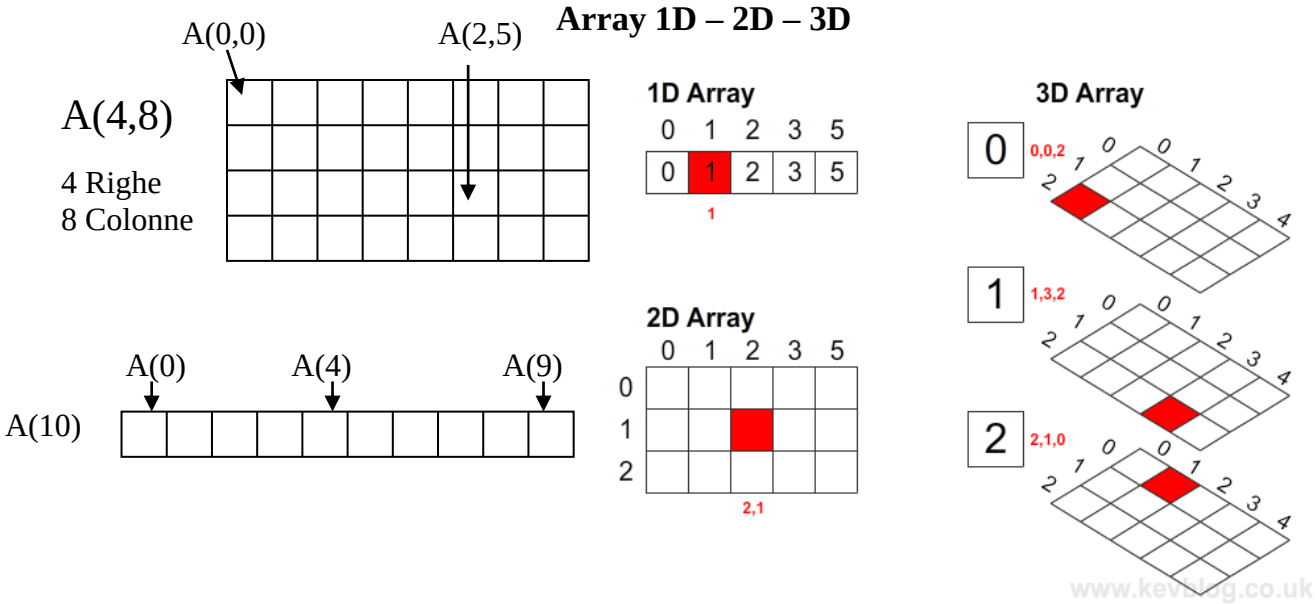


Le singole caselle vengono identificate tramite un’altra variabile – detta indice

CUBLOC BASIC supporta array fino a 8 dimensioni, ogni dimensione può contenere massimo 65535 caselle.
Per la dichiarazione degli array si usa la stessa metodologia della dichiarazione delle singole variabili, si deve specificare, tramite l’uso di parentesi, il numero delle caselle.

Esempio

Dim A(6) As byte ‘Dichiara un Array (vettore) composto da 6 caselle di tipo Byte
Dim A(3,6) As integer ‘Dichiara un Array 2D (matrice) composto da 3 righe e 6 colonne di tipo Integer.
Dim T(3,4,6) As Single ‘Dichiara un Array 3D(Matrice) composto da 3 piani, 4 righe e 6 colonne di tipo single.



1.5.4 Le costanti

Costanti vengono utilizzate per dichiarare un valore fisso all'interno del programma, questo permette di assegnare un numero ad un nome.
Nel basic del Cubloc è possibile definire le seguenti costanti:

- Intere: numeri senza parte frazionaria (esempio 45)
- Reali: numeri formati da una parte intera e da una parte frazionaria separate dal punto (esempio 34.65)
- Stringa: costituite da una sequenza di caratteri racchiusi tra doppi apici (ad esempio "Controllo temperatura")
- Binaria: la sequenza di bit deve essere preceduta da &B oppure 0b (Esempio: &B10011110, 0b00011101)
- Esadecimale: la sequenza di simboli deve essere preceduta da &H,0x oppure $ (Esempio: &HABF0, 0xB0E4, $C6AD)

È possibile, all'inizio del programma, definire delle costanti personalizzate.
Per la definire le costanti si usa il comando CONST oppure CON

Comando CONST (sintassi ed esempi)
CONST *identificatore* As *tipo= valore*

dove *identificatore* è un nome, *tipo* (byte, integer, long, single, string), *valore* rappresenta il dato da assegnare al nome.Se il tipo non viene specificato, il compilatore associa alla costante uno appropriato

CONST PI AS SINGLE=3.14159
CONST WR AS BYTE=10
CONST MSG AS STRING="Temperatura max"

```
CONST  PI = 3.14159            ' Declare as SINGLE
CONST  WRTTIME  = 10           ' Declare as Byte
CONST  MYROOM  = 310           ' Declare as Integer since it's over 255.
CONST  MSG1 = "ACCESS PORT"    ' Declare as String
```

Comando CON (sintassi ed esempi)
identificatore CON *valore*

dove *identificatore* è un nome, *valore* rappresenta il dato da assegnare al nome.
Se il tipo non viene specificato, il compilatore associa alla costante uno appropriato.

```
PI            CON    3.14159       ' Declare as SINGLE.
WRTTIME  CON    10            ' Declare as Byte
MYROOM   CON    310                 ' Declare as Integer
MSG1          CON    "ACCESS PORT"       ' Declare as String
```

Array di costanti
È possibile definire array di costanti racchiudendo i valori, separati da virgole, tra una coppia di parentesi:

CONST *Byte DATA = (31,25,102,34)*

L'accesso al generico elemento avviene, facendo riferimento all'indice, a partire da zero;
ad esempio DATA(0) è pari a 31, DATA(l) è pari a 25

1.5.5 Operatori ed espressioni

Il BASIC CUBLOC definisce quattro classi di operatori:

1. Aritmetici
2. Logici
3. Relazionali
4. Bit

Operatori aritmetici
Gli operatori aritmetici supportati sono:

1. + addizione
2. - sottrazione
3. * Moltiplicazione
4. / Divisione
5. Mod Resto della divisione
6. ^ Elevazione a potenza
7. Decr Decremento di uno del contenuto di una variabile intera
8. Incr Incremento di uno del contenuto di una variabile intera

Operatori relazionali
Gli operatori relazionali hanno la funzione di confrontare due valori (o espressioni). Il risultato di una operazione relazionale è una condizione logica (vero o falso)
Gli operatori relazionali supportati sono:

1. > maggiore
2. >= maggiore o uguale
3. < minore
4. <= minore o uguale
5. = uguale
6. <> diverso

Operatori logici
Gli operatori logici permettono di confrontare condizioni logiche, restituendo un'ulteriore condizione logica.
Gli operatori logici supportati :

1. AND Restituisce la condizione vero soltanto quando entrambi gli operandi sono vero
2. OR Restituisce la condizione vero soltanto quando almeno uno dei due operandi è vero
3. NOT Restituisce vero se l'operando è falso, restituisce falso se l'operando e vero
4. XOR restituisce vero soltanto quando i suoi operandi hanno valore logico differente.

Operatori sui bit

Gli operatori sui bit consentono di manipolare i singoli bit che compongono un byte o una word (2 byte): Gli operatori sui bit supportati :

1. AND
2. OR
3. NOT
4. XOR

Si osservi, però, che stavolta questi operatori lavorano a livello dei singoli bit, quindi se, per esempio, scriviamo l'espressione &H33 AND &HCC, il risultato sarà calcolato mediante l'AND bit a bit dei numeri binari 00110011 e 11001100, che dà il valore 00000000, che corrisponde al numero esadecimale &H00.

Tutti gli operatori esaminati possono essere combinati con costanti e variabili per costruire espressioni. Una cosa di cui bisogna tener conto, in primo luogo, nella costruzione delle espressioni, è la priorità degli operatori nella valutazione, che comunque può essere modificata facendo uso delle parentesi tonde.

$\frac{1}{2} \Longrightarrow 1/2$

$\frac{5}{3+4} \Longrightarrow 5 / (3+4)$

$\frac{2+6}{3+4} \Longrightarrow (2+6) / (3+4)$

La massima precedenza è assegnata alle parentesi, seguita in ordine dal meno unario, dall'elevazione a potenza, dalla moltiplicazione/divisione/resto, dall' addizione/sottrazione, dallo shift a sinistra/destra, dagli operatori relazionali, dagli operatori logici sui bit, dagli operatori logici ed infine dall'assegnazione.

In secondo luogo bisogna osservare che, se nelle espressioni sono utilizzati dati di tipi differenti, questi vengono convertiti tutti nel tipo che occupa più memoria fra quelli utilizzati (promozione di tipo), Nell'operazione di assegnazione, il valore alla sinistra del segno di uguale viene invece convertito nel tipo di destra e quindi si può verificare la perdita dell'informazione. Se, ad esempio, scriviamo l'espressione seguente:

```
Dim FI As Single
FI = 3/4
```

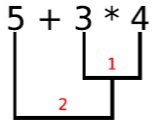
noteremo che la nostra variabile FI non vale 0.75, come ci aspettavamo, bensì 0. Infatti in un caso del genere la divisione viene forzata al valore intero e si ha perdita di informazione; il codice corretto è il seguente:

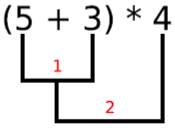
```
Dim FI As Single
FI = 3.0/4.0
```

Priorità Operatori

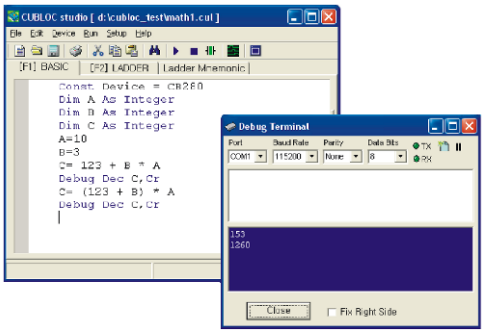
When multiple operators are used, the following operator priorità is used:

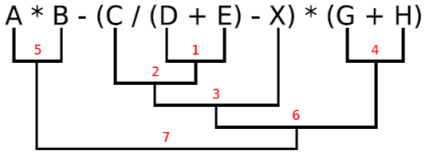
1) Operator inside parenthesis
2) Negative Sign (-)
3) Exponent (^)
4) Multiplication, Division, Remainder (*, /, MOD)
5) Addition/Subtraction (+,-)
6) Bitwise Left Shift, Bitwise Right Shift (<<, >>)





Demo Program





1.6 Strutture ed Istruzioni di controllo

Nella programmazione Basic Cubloc sono previste diverse strutture di controllo, con la premessa che per blocco intendiamo una sequenza di operazioni.

1.6.1 Struttura selezione semplice

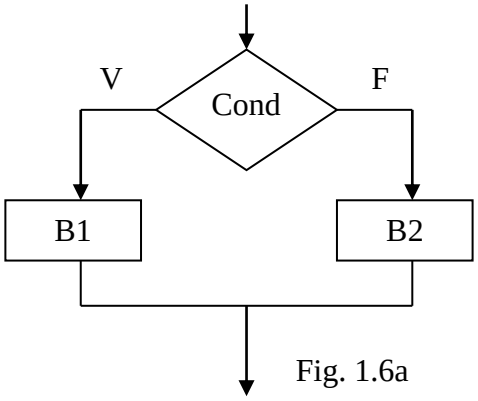


Fig. 1.6a

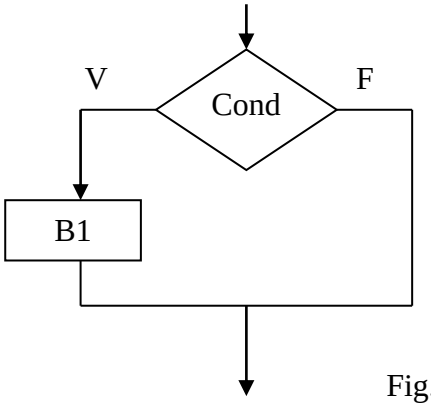
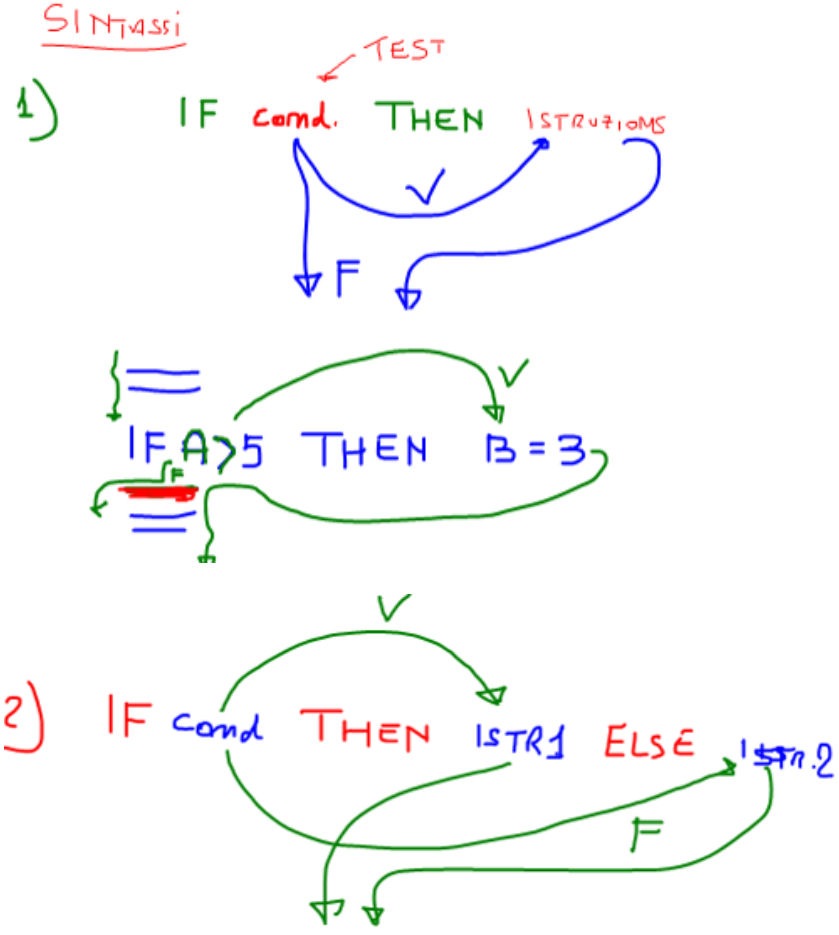


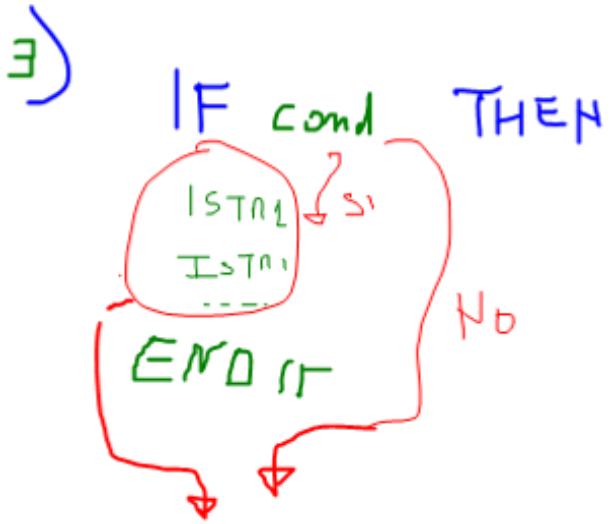
Fig. 1.6b

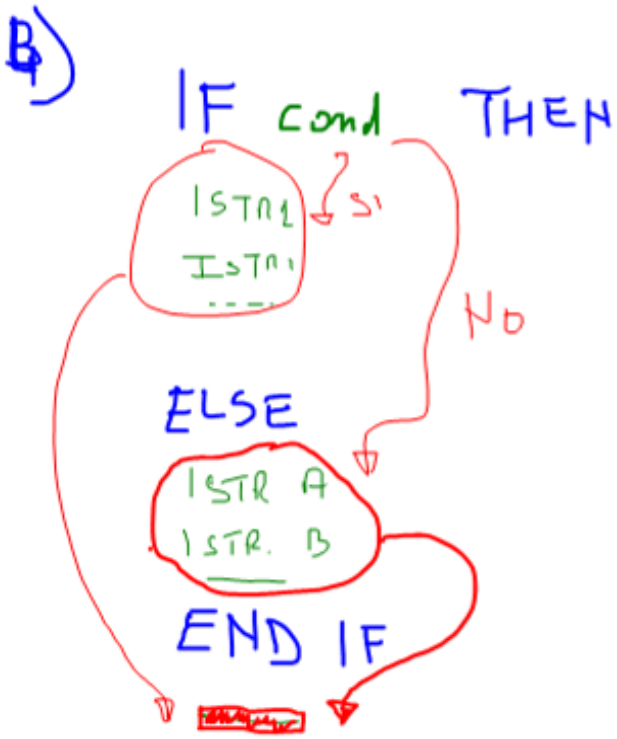
Un'istruzione di selezione semplice (istruzione condizionale) determina l'esecuzione delle operazioni di un blocco B1 (se la condizione è Vera), oppure l'esecuzione del blocco B1 (se la condizione è Falsa) (Fig.1.6a). La struttura può anche non prevedere un'alternativa (Fig. 1.6b). Il blocco *Cond* è costituito da un'espressione con operatori relazionali (es. A>=45). Nella stesura del programma (listato) la struttura viene tradotta con l'istruzione:

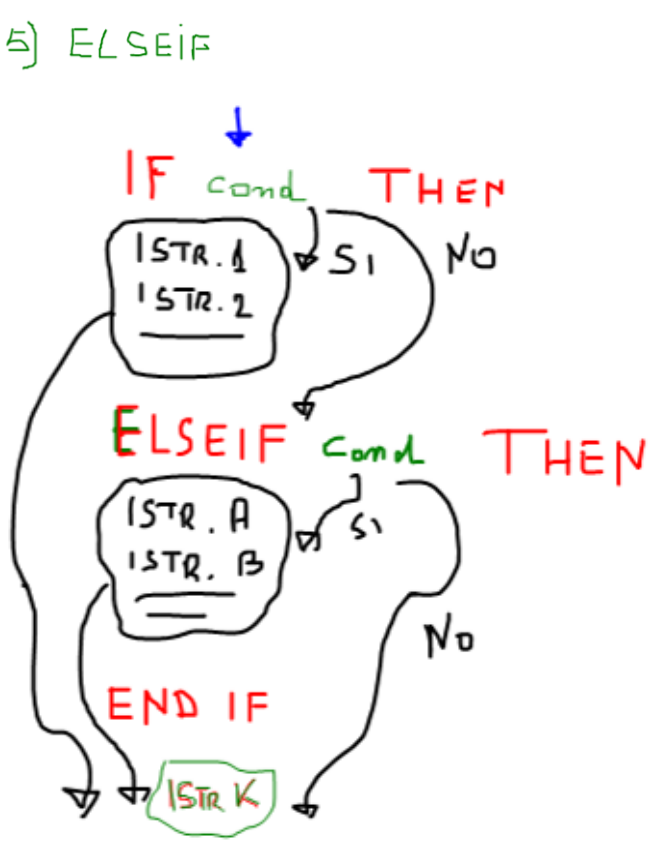
IF … Then … ElseIf …Endif

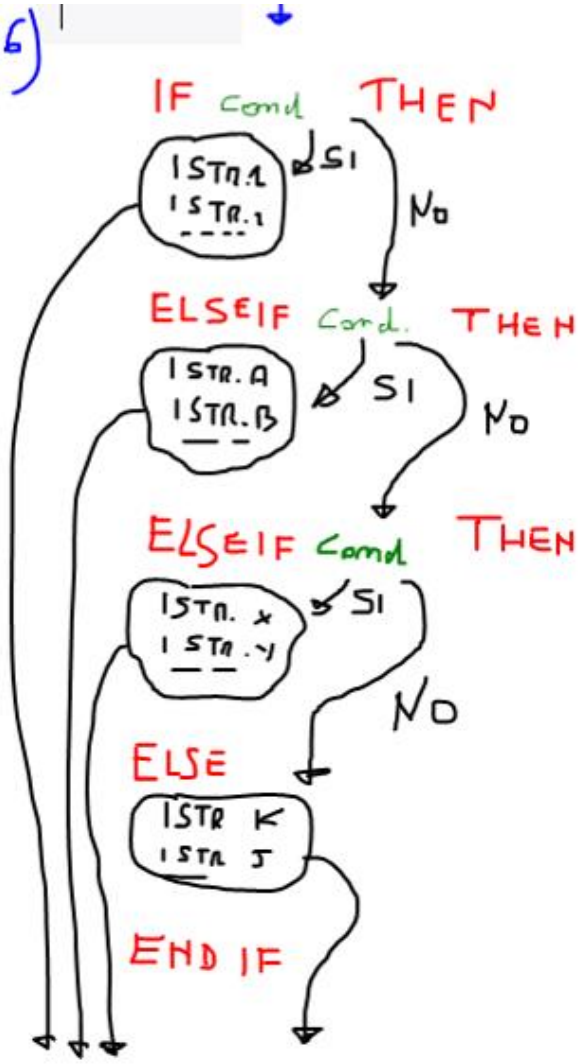
Nel Basic Cubloc esistono 6 alternative:




3)
IF cond THEN
ISTR1
ISTR2
SI
NO
END IF


4)
IF cond THEN
ISTR1
ISTR2
SI
NO
ELSE
ISTR A
ISTR. B
END IF


5) ELSEIF
IF cond THEN
ISTR. 1
ISTR. 2
SI
NO
ELSEIF cond THEN
ISTR. A
ISTR. B
SI
NO
END IF
ISTR K


6)
IF cond THEN
ISTR.1
ISTR.2
SI
NO
ELSEIF Cond. THEN
ISTR. A
ISTR. B
SI
NO
ELSEIF cond THEN
ISTR. x
ISTR. y
SI
NO
ELSE
ISTR K
ISTR J
END IF

1.6.2 Struttura selezione multipla

La struttura selezione multipla viene utilizzata quando in base al valore assunto da una variabile debbano essere prese decisioni diversificate.

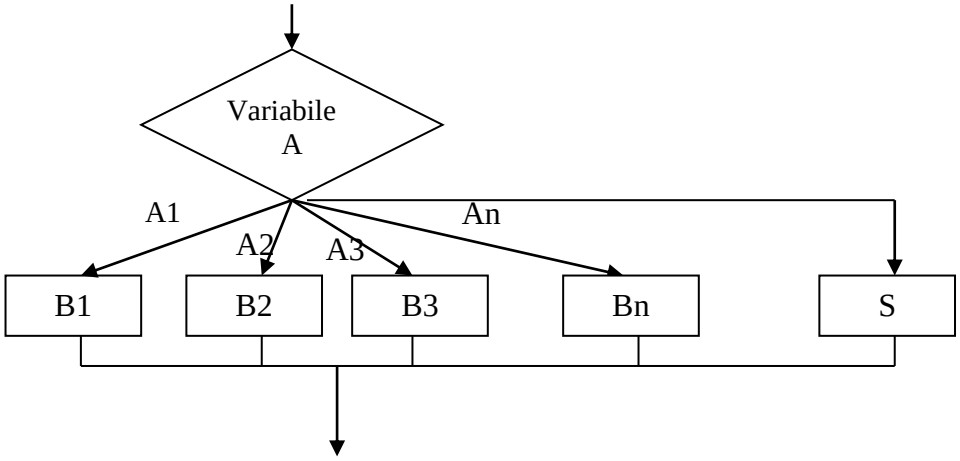


Fig. 1.6c

L'esecuzione di uno dei blocchi alternativi B1, B2, Bn dipende da una variabile di controllo (Es. A), che può assumere uno qualunque dei valori dell'insieme {A1, A2, A3, ..An}.
Se la variabile A non assume alcuno dei valori del predetto insieme, viene eseguito il blocco chiamato S (Fig. 1.6c)
Nella stesura del programma (listato) la struttura viene tradotta con l'istruzione:

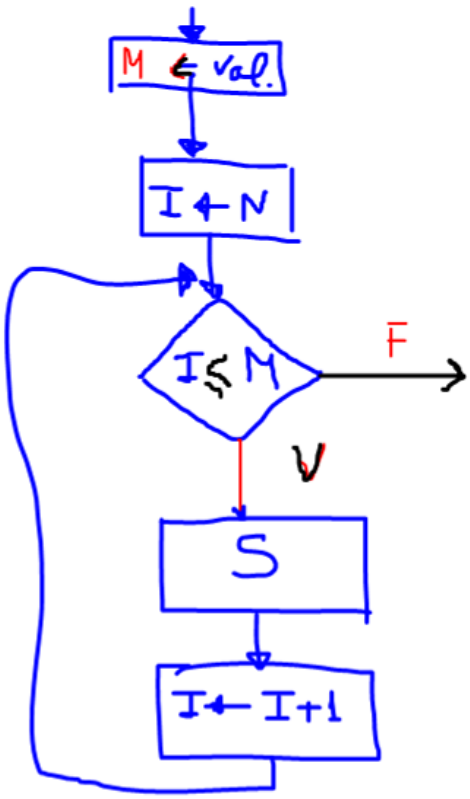
SELECT … CASE

La forma sintattica della struttura decisionale Select .. Case è la seguente:
If the condition Value of Case is met, the Statement under the case is executed.

```
Select Case Variable
          [Case Value [,Value],…
                [Statement 1]]
          [Case Value [,Value],…
                [Statement 2]]
          [Case Else
                [Statement 3]]
End Select
```

```
Select Case A
          Case 1
                B = 0
          Case 2
                B = 2
          Case 3,4,5,6          ' Use Comma(,) for more than 1 value.
                B = 3
          Case Is < 1           ' Use < for logical operations.
                B = 3
          Case Else             ' Use ELSE for all other cases.
                B = 4
End Select

Select Case K
          Case  Is < 10         ' If less than 10
                R = 0
          Case Is < 40          ' If less than 40
                R = 1
          Case Is < 80
                R = 2
          Case Is < 100
                R = 3
          Case Else
                R = 4
End select
```

1.6.3 Struttura iterazione enumerativa



Si fa ricorso a questa struttura (diagramma qui a lato) quando è noto il numero **N** di iterazioni che l'esecutore deve svolgere.
La variabile di controllo **I** all'ingresso del ciclo riceve il numero **N** (*Valore iniziale*) e ogni volta prima di eseguire il ciclo, si confronta il valore **I** con quello finale **M**.
Se **I≤M** si entra nel ciclo e la variabile **I** viene incrementata di un'unità.
Quando il valore supera **M**, il ciclo non viene eseguito e l'iterazione ha termine.
Il blocco **S** rappresenta una sequenza di istruzioni.
Nella stesura del programma (listato) la struttura viene tradotta con l'istruzione:

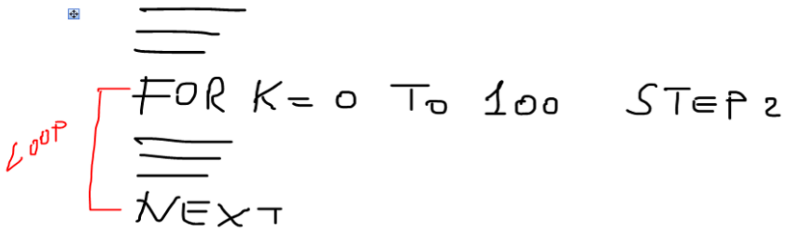
For .. Next

La forma sintattica dell'istruzione è la seguente:

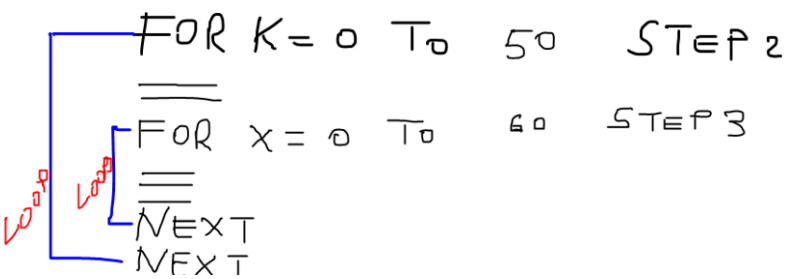
For *Variabile=valore iniziale* **To** *Valore finale* **Step** *Valore Incremento*
……...
……
Next

La *Variabile* viene inizializzata con il valore di partenza *valore iniziale* ed è incrementata della quantità *Valore Incremento* finché non ha raggiunto il suo *Valore finale*.
Il valore dello Step può essere anche negativo; se omesso viene assunto pari ad uno.
La clausola *Exit For* consente di uscire anticipatamente dal loop.
Quest'istruzione viene in genere utilizzata nella gestione degl'indici degli array.

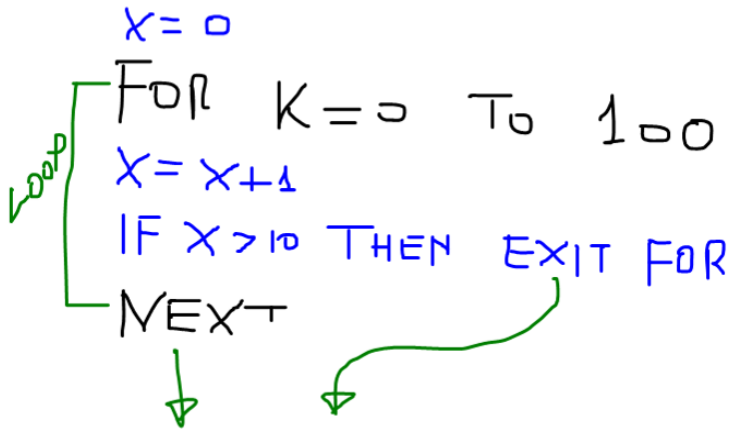
Esempio 1 – Loop da 0 a 100 con incremento di 2



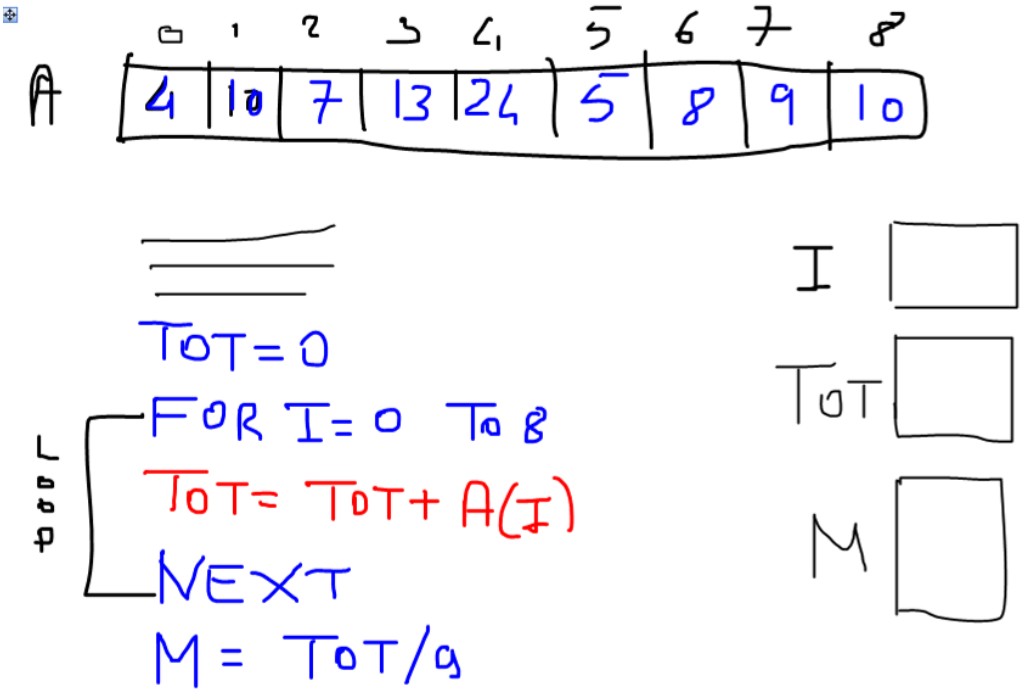
Esempio 2 – Loop nidificati (Interno da 0 a 60 con Step 3) (Esterno da 0 a 50 con Step 2)



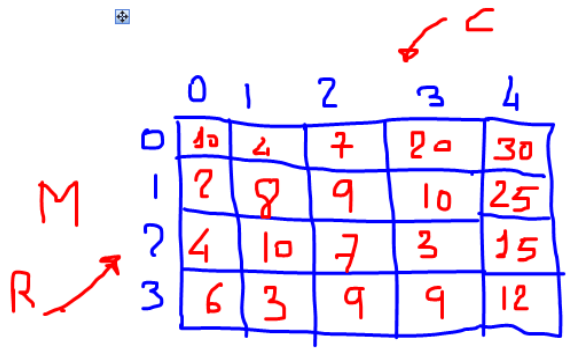
Esempio 3 – Loop con la clausola Exit For



Esempio 4 – assegnato l'array di figura calcolare il totale e la media dei valori

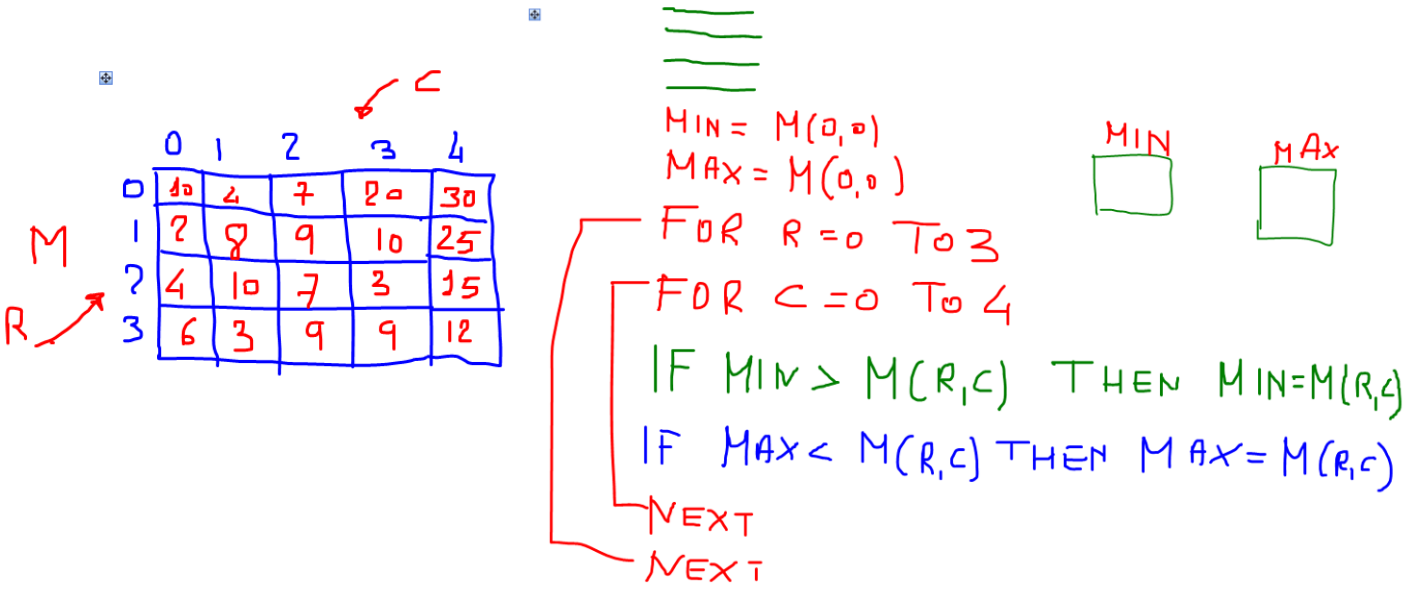


Esempio 5 – assegnato l'array di figura calcolare il totale e la media dei valori

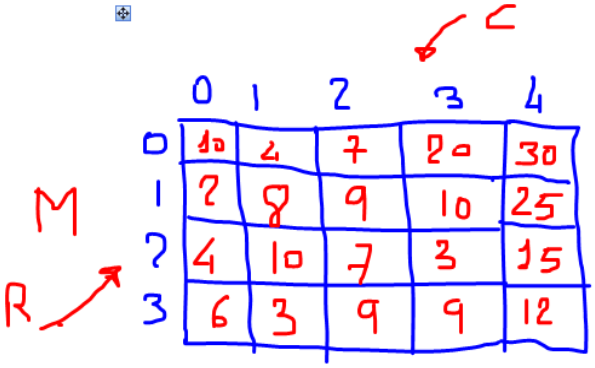


……
……
Tot=0
For R=0 To 3
For C=0 To 4
Tot=Tot+M(R,C)
Next
Next
M=Tot/20
……

Esempio 6 – assegnato l'array di figura trovare il valore minimo ed il valore massimo



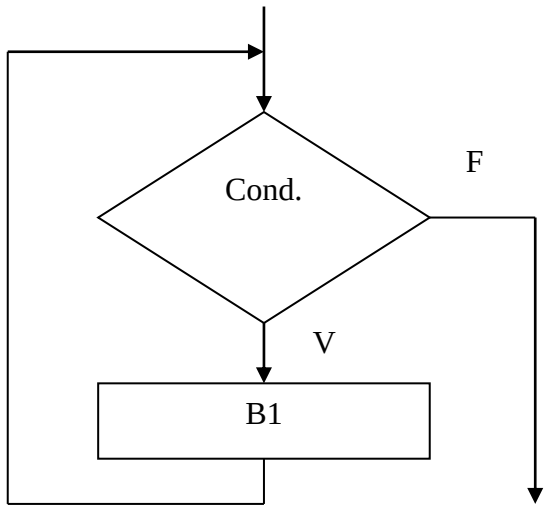
Esempio 7 – assegnato l'array di figura determinare quante volte si ripete il numero 10



……
……
N=0
For R=0 To 3
For C=0 To 4
IF M(R,C)=10 Then N=N+1
Next
……

1.6.4 Struttura iterazione a controllo iniziale

Un blocco di operazioni viene eseguito una o più volte mentre la condizione di controllo, posta all'inizio del ciclo, è vera. Quando la condizione è falsa si esce dal ciclo. Se la condizione di controllo è sempre falsa, il blocco **B1** non viene mai eseguito.



Finché condizione *vera* ➔ Esegui

La forma sintattica della struttura iterazione a controllo iniziale è la seguente:

Do While condizione logica
……
……
Loop

Condizione logica:
Espressione con operatori relazionali
Es. A>25

Se si omette la clausola While il loop (ciclo) viene eseguito all'infinito
La clausola Exit Do consente di uscire anticipatamente dal loop

Esempio 1

```
X=0
Do While X<10
Incr x
Loop
```

← Esegue il Loop finché X<10

Esempio 2

```
…….
Do
…….
…….
Loop
```

← Loop all'infinito

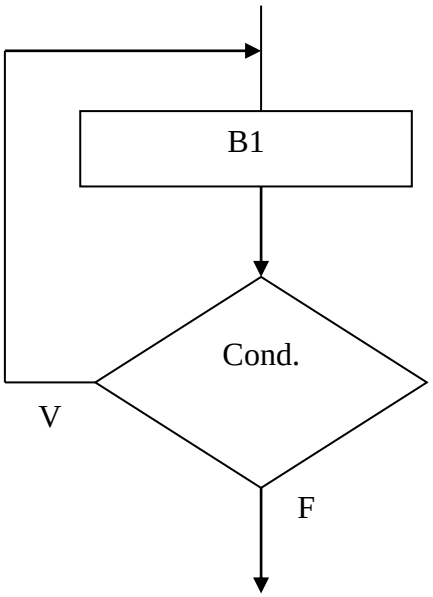
Esempio 3

```
A=0
Do
A=A+1
If A>=20 Then Exit Do
…….
Loop
```

← Loop con la clausola Exit Do

1.6.5 Struttura iterazione a controllo finale

Il blocco d'istruzioni corrispondente viene eseguito ripetutamente fino a quando la condizione di controllo, posta alla fine del ciclo, da vera diventa falsa. Il blocco **B1** viene sempre eseguito almeno una volta.



Ripeti B1 finché condizione *vera*

La forma sintattica della struttura iterazione a controllo finale è la seguente:

```
Do
……
……
Loop While  condizione logica
```

Condizione logica:
Espressione con operatori relazionali
Es. B>5

Se si omette la clausola While il loop (ciclo) viene eseguito all'infinito
La clausola Exit Do consente di uscire anticipatamente dal loop

Esempio 1

```
X=0
Do
Incr x
Loop While X<10
```

← Ripeti finché X<10

Esempio 2

```
…….
Do
……..
……..
Loop
```

← Loop all'infinito

Esempio 3

```
A=0
Do
A=A+1
If A>=20 Then Exit Do
…….
Loop While X<10
```

← Loop con la clausola Exit Do

1.7 Sottoprogrammi (Subroutine)

Nel Basic Cubloc l'utente, al fine di scomporre e semplificare problemi complessi, può definire sottoprogrammi, che devono essere obbligatoriamente posizionati al termine del programma principale (se questi vengono utilizzati, il programma principale deve essere terminato con la clausola **End**.
La struttura dei sottoprogrammi è la seguente:

Sub *nome_sottoprogramma* (*elenco argomenti*)
    sequenza di istruzioni
End Sub

Dove *elenco argomenti* è un elenco di nomi di variabili con specificato il tipo corrispondente (Byte, Integer, single, Long, String), separati dalla virgola, che quando viene chiamato il sottoprogramma ricevono i valori e li trasferiscono a quest'ultimo.
Le parentesi tonde vanno messe anche se non sono presenti argomenti.
La clausola opzionale **exit sub** consente di uscire anticipatamente dalla subroutine.
Nel Flow Chart del programma principale (main program) la chiamata della subroutine viene rappresentata con il simbolo riportato in Fig. 1.7a, in Fig. 1.7b è riportato il Flow Chart di una generica subroutine, in Fig. 1.7c è riportata la struttura del programma principale con la chiamata alla subroutine ed il relativo flusso dell'esecuzione.

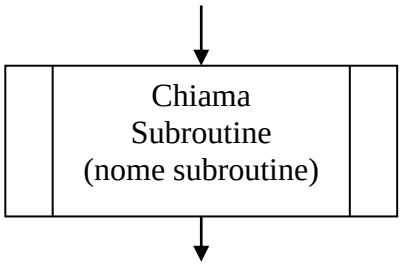


Fig. 1.7a

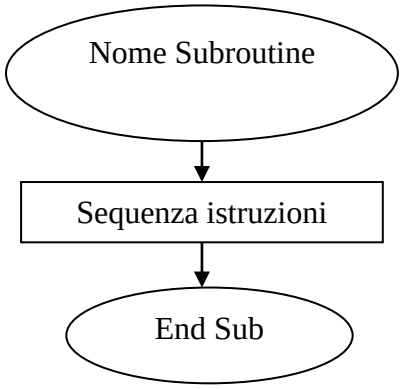


Fig. 1.7b

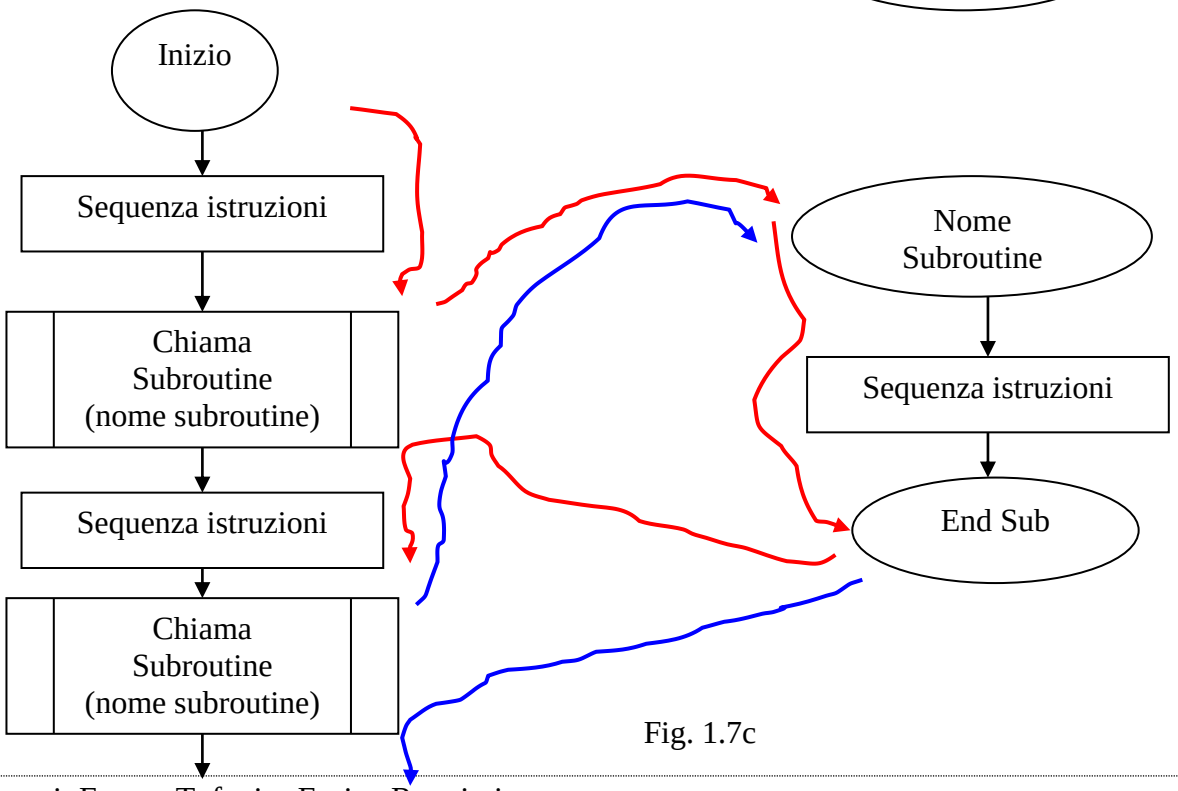


Fig. 1.7c

Esempio 1 – Chiamata subroutine con passaggio di valori

```
------------
------------
------------
Conteggio 600
------------
------------
End

Sub conteggio (DX as integer)
        Dim K as integer
                For K= 0 to DX
        Next
End Sub
```

Il comando **conteggio 600** chiama la subroutine di nome **conteggio** e memorizza nella variabile **DX** il valore **600**.
La subroutine esegue il loop da 0 a 600. il comando end sub permette il ritorno al programma principale

Esempio 2 - Chiamata subroutine senza passaggio di valori

```
------------
------------
------------
Conteggio
------------
------------
End

Sub conteggio ()
        Dim K as integer
                For K= 0 to 600
        Next
End Sub
```

Il comando **conteggio** chiama la subroutine di nome **conteggio**.
La subroutine esegue il loop da 0 a 600.
Ill comando end sub permette il ritorno al programma principale

Esempio 3 - Chiamata subroutine con passaggio di valori

```
------------
------------
------------
Conteggio 500, 800
------------
------------
End

Sub conteggio (L as integer, P as integer)
        Dim R as integer
        Dim C as integer
                For K= 0 to L
                For C=o to P
        Next
        Next
End Sub
```

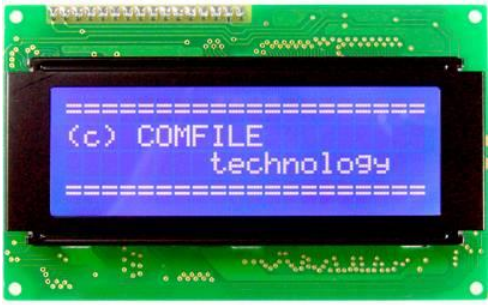
Il comando **conteggio 500, 800** chiama la subroutine di nome **conteggio**.
Memorizza il valore 500 nella variabile L ed il valore 800 nella variabile P
La subroutine esegue il doppio loop con i valori indicati.
Il comando end sub permette il ritorno al programma principale

1.8 Visualizzazione - Display numerici, testuali CLCD e grafici

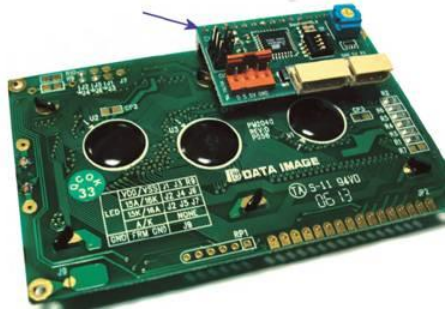
Per la visualizzazione delle informazioni la COMFILE mette a disposizione diversi tipi di display.

1.8.1 Display alfanumerici CLCD

I display CLCD sono pilotabili direttamente dal Cubloc, senza aggiunta di altre interfacce. Disponibili in diversi modelli, essi riescono a soddisfare praticamente qualsiasi esigenza dell'utente.



Il CLCD è un display LCD retroilluminato, in grado di visualizzare lettere, numeri e simboli.
Nella parte posteriore del modulo è presente un piccolo controller che riceve i dati dal Cubloc e li visualizza direttamente sul display.



il collegamento elettrico

Il display CLCD prevede due tipologie di collegamento, supportate naturalmente anche dai relativi comandi Basic. I metodi di trasmissione sono i seguenti:

- **metodo 0**, che adotta il protocollo CuNET;
- **metodo 1**, che utilizza il canale 1 RS232 del Cubloc.

Metodo 0 – Uso del CuNET (fig 1.8.1a), il CuNET è un protocollo simile all'I2C, gestito perfettamente dal Cubloc. Almeno per i modelli Cubloc CB220 e CB280, tale protocollo, che utilizza solamente due linee di trasmissione, fa capo alle porte di comunicazione **P8** (segnale di clock) e **P9** (segnale di trasporto dati). Esso non gestisce la configurazione del proprio Baud Rate, dal momento che la velocità di trasmissione è fissa.
Metodo 1 – Uso del Canale 1 RS232 Per poter utilizzare i moduli display, occorre utilizzare il Canale 1 RS232 del Cubloc. Per il modello Cubloc CB220, la linea TX fa capo alla porta P11, mentre la linea RX fa capo alla porta P10. Per il modello Cubloc CB280, la linea TX fa capo sia al pin 33 (con livello di segnale +/- 12V) che al pin 49 (con livello di segnale TTL 0/5V), la linea RX fa capo sia al pin 34 (con livello di segnale +/- 12V) che al pin 50 (con livello di segnale TTL 0/5V).
Utilizzando questo tipo di collegamento, la massima velocità di trasmissione, con potenziale di +/- 12V è di 38400 bps, mentre con un potenziale di 5V TTL può arrivare fino 240400 bps.

Il controller del modulo CLCD

Sul lato posteriore del modulo CLCD si trova il controller di comunicazione. La sua funzione è quella di accettare, interpretare e, quindi, inviare all'unita LCD, le informazioni.
Esso può ricevere i dati utilizzando il protocollo CuNET ovvero la comunicazione RS232. Vi sono infatti due connettori adatti a tale tipo di trasmissione:
uno che utilizza 3 pin con una tensione di segnale TTL di 5V, l'altro che utilizza 4 pin con una tensione standard di +/-12V. Il controller può essere facilmente rimosso dall'unità.
E' presente inoltre un DIP switch che ha la funzione di configurare la velocità del modulo (nel caso di protocollo RS232) o l'indirizzo Slave (nel caso di protocollo I2C o CuNET).

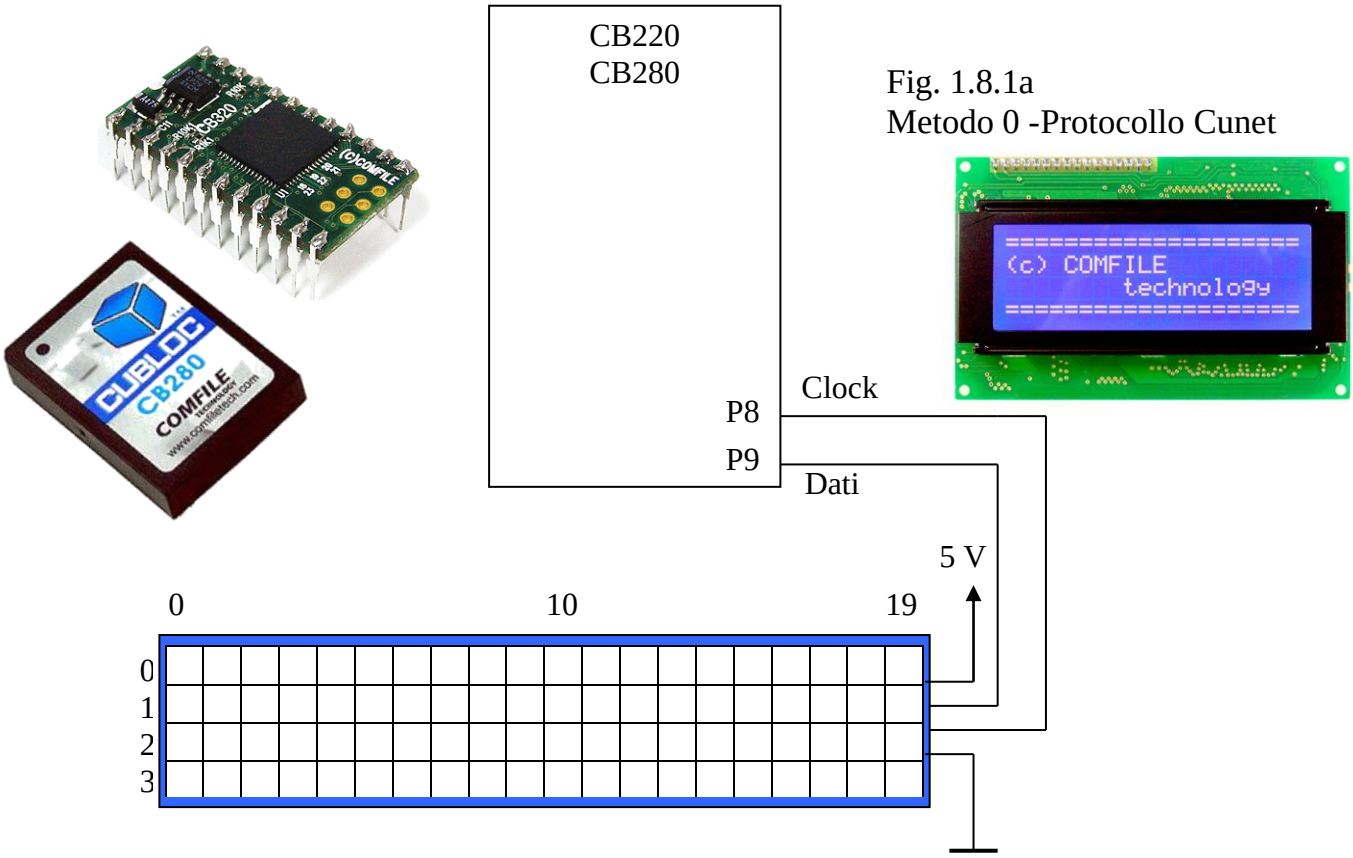


Fig. 1.8.1a
Metodo 0 -Protocollo Cunet

Lo schermo indirizzabile

Come detto in precedenza, la Compile ha prodotto due serie di display CLCD, diverse tra loro per la dimensione indirizzabile dello schermo, cioè per il numero di righe e colonne presenti su ciascuno di essi. I due tipi di schermo prevedono i seguenti numeri di riga e colonna:

• **20 colonne x 4 righe -** (modelli CLCD420-B e CLCD420-G);
• **16 colonne x 2 righe –** (modello CLCD216-G).

La prima colonna è localizzata con la coordinata 0, la prima riga con la coordinata 0.
Le successive seguono progressivamente tale numero.

I COMANDI BASIC

Il Cubloc gestisce comandi ad alto livello, in linguaggio Basic.
Con essi è praticamente possibile realizzare complessi prototipi, utilizzando un numero minimo di istruzioni. Di seguito è riportato l'elenco dei comandi disponibili, con la relativa sintassi ed esempi di utilizzo.

SET DISPLAY

Questo comando è utilizzato per configurare il modulo display. Va usato prima delle altre istruzioni di visualizzazione su LCD e, in ogni caso, una sola volta.
La sintassi è la seguente:

SET DISPLAY *type, method, baud, buffersize*

I quattro parametri hanno le seguenti funzioni:

type 0=Rs232LCD, 1=GHLCD, 2=CLCD;

method 0=CuNET, 1=COM1;

baud velocità di trasmissione da impostare (nel caso si preveda il collegamento RS2332) oppure l'indirizzo slave (nel caso si preveda il collegamento CuNET). I valori di Baud Rate consentiti sono i seguenti: 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200, 230400.

Buffersize specifica la grandezza del buffer. Si raccomanda di utilizzare valori compresi tra 50 e 128. Se si specifica un valore troppo basso, la visualizzazione non avviene correttamente, mentre se si specifica un valore troppo alto, si rischia di "ingolfare" la memoria del display. Esempio di utilizzo:

SET DISPLAY 0,1,19200,50

Questo esempio configura un display di tipo RS232 con velocità di trasmissione pari a 19200 e ampiezza buffer di 50.
Altro esempio:

SET DISPLAY 2,0,0,50

Il comando configura un display CLCD con protocollo CuNET, con indirizzo slave pari a 0 e buffer di 50.

Cls

Questo comando (Clear Screen) è utilizzato per cancellare il contenuto del display, cioè per rimuovere tutti i caratteri presenti. Dal momento che la sua funzionalità è relativamente lenta, è consigliabile inserire una piccola pausa d'attesa dopo la sua esecuzione.
Esempio di utilizzo:

```
CLS
DELAY 200
```

Csron

Serve per attivare la visualizzazione del cursore.

Csroff

Disattiva la visualizzazione del cursore.

Locate

E' uno dei comandi fondamentali per la gestione del display. Serve per posizionare il cursore alle varie coordinate, per la successiva visualizzazione dei caratteri.
La sintassi è la seguente:

LOCATE X,Y
dove X rappresenta la coordinata di colonna e Y quella di riga.
Print
E' il comando più utilizzato, in quanto invia direttamente sul display i caratteri da visualizzare. Dietro questa semplice istruzione, sono nascoste routine estremamente complesse, a basso livello, che indirizzano i singoli registri del controller e del display.
La sintassi è la seguente:
PRINT stringa/variabile

Esempio di utilizzo:
LOCATE 0,2
PRINT "Fare Elettronica"

VISUALIZZARE LE VARIABILI
Se la visualizzazione delle stringhe di testo non comporta problemi di sorta, quella delle variabili necessita di una variazione sulla programmazione.
Se infatti predisponessimo normalmente l'istruzione PRINT per lo scopo, verrebbe stampato il carattere corrispondente al suo codice ASCII e non il valore.
Un esempio chiarirà il concetto.
DIM K AS BYTE
K=65
PRINT K
In questo esempio, dopo il dimensionamento della variabile K quale tipo byte (0-255) e la relativa assegnazione del valore decimale 65, l'istruzione di visualizzazione PRINT non produrrà sul display l'atteso numero 65, bensì il corrispondente carattere ASCII, cioè la lettera 'A'. Questo avviene poiché l'istruzione considera i suoi parametri come byte, indipendentemente dalla loro forma numerica o stringa.

Se si vuole visualizzare il reale valore della variabile, basterà rettificare il comando con il seguente:
PRINT DEC K

In questo caso, il contenuto della variabile K viene convertito in un numero decimale a 10 bit. Utilizzate tale metodo per visualizzare il contenuto delle vostre variabili.
Il Cubloc prevede altri tipi di operatori di conversione. I modificatori HEX e DEC accettano anche le relative varianti (HEX1, HEX2, ecc. oppure DEC1, DEC2, ecc.) per "fissare" il numero di cifre decimali utili alla visualizzazione.

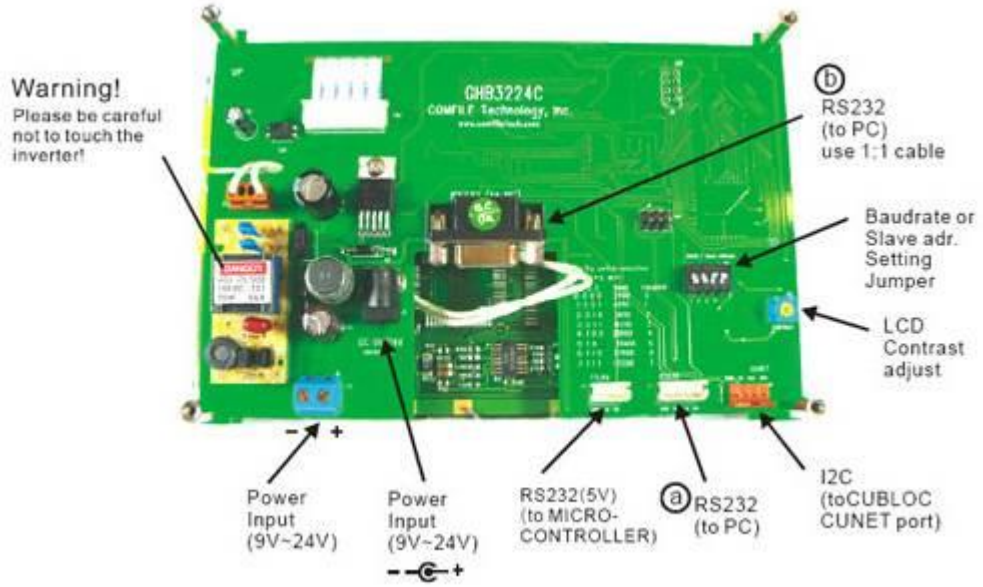
Esempio combinato stringa numero
PRINT "Temp=",DEC K,"°C"

Esempio

```
Const Device=CB280

Set Display 2,0,0,128

Cls    'Clear screen
Delay 200
Csroff 'Cursor Off

Locate 0,0
Print "====================="
Locate 0,1
Print "(c) COMFILE"
Locate 8,2
Print "technology"
Locate 0,3
Print "====================="
```

Display LCD grafico blu e bianco 320x240 con retroilluminazione

Il GHB3224C è un display LCD grafico con retroilluminazione CCFL. Possibilità di creare caratteri personalizzati, visualizzare e memorizzare i file BMP utilizzando la memoria flash del display LCD. Dispone inoltre di punti, linee, caselle e cerchi utili per creare applicazioni GUI friendly. Per l''invio dei dati l''utente può utilizzare un CUBLOC (utilizzando il CUNET), un PC o un microcontrollore.

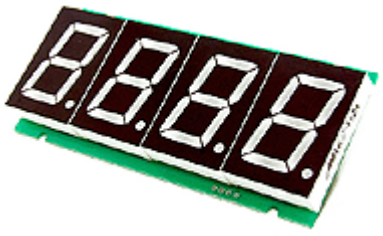


SPECIFICHE

- Supporta i comandi: dot, line, circle, ellipse, box, pop, paste e paint.
- Baud rates: 2400 ~ 115.200.
- Interfacce supportate: I2C, RS232C (12 V) e RS232 (5V).
- Strati separati per testo e grafica (totale di 3 strati).
- Lo strato testo supporta: scroll, cursor, inversion, underline, and bold.
- Lo strato testo utilizza caratteri basati su pixel e lo strato grafica utilizza Dot basati su pixel.
- Supporta i comandi push / pop / paste (come copia e incolla in Windows).
- Visualizza file BMP.
- Imposta la dimensione per Lines, Circles, Boxes, Ellipses.
- Retroilluminazione e contrasto controllati via software.
- Casella di controllo del programma per testare il display LCD.
- 4 Built-in Fonts.

DOWNLOAD

- La documentazione tecnica completa, i manuali hardware e software nonchè tutto il software necessario per fare funzionare i moduli Cubloc può essere scaricato gratuitamente dal sito del produttore: www.cubloc.com.

Display Numerici a 7 segmenti	
Modello	**Descrizione**
CSG-4M/S (Medium/Small) 	**Display composto da 4 cifre a 7 segmenti** Protocollo:CUNET (I2C)

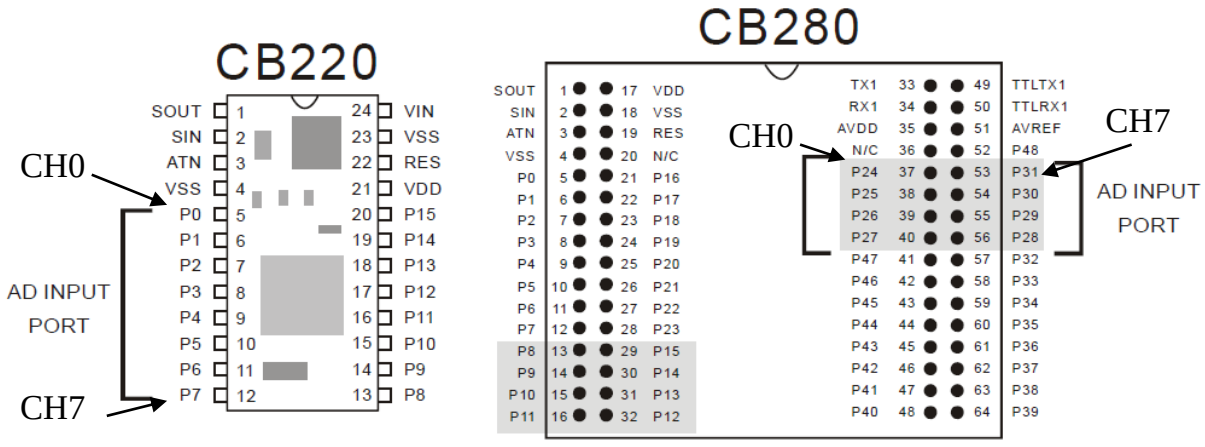
1.9 Convertitore A/D

I moduli Cubloc dispongono al proprio interno un convertitore Analogico/Digitale a 10 bit ed 8 canali che consente di collegare direttamente al chip dispositivi esterni, i quali generano segnali che abbiano un'escursione di tensione compresa tra 0V÷5V.
Il segnale analogico viene letto attraverso un piedino di I/O abilitato al funzionamento analogico, convertito in un numero digitale a 10 bit e memorizzato all'interno della memoria RAM (Variabile/Array).

Il convertitore è caratterizzato dai seguenti parametri:

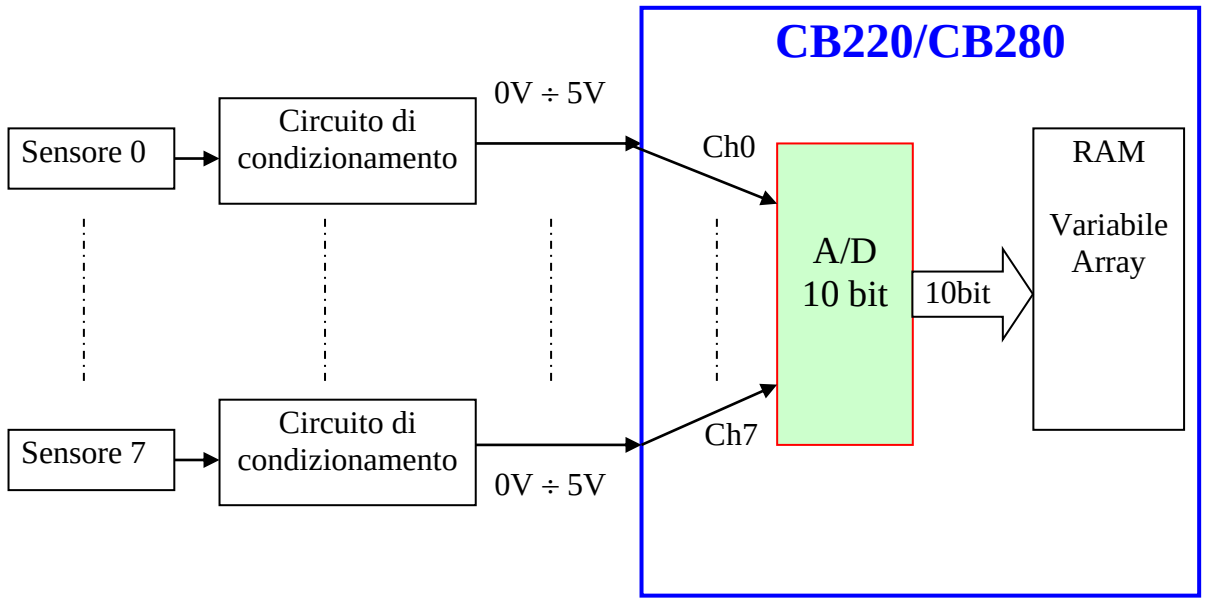
1) Risoluzione digitale 10 bit
2) VFS (tensione fondo scala)=5V
3) Range di Input=0V÷5V
4) Range di output=0÷1023
5) Risoluzione analogica $q = \frac{Vfs}{2^{10}} = \frac{5-0}{1024} = 4{,}88mV$

In figura sono evidenziati i pin abilitati al funzionamento analogico.

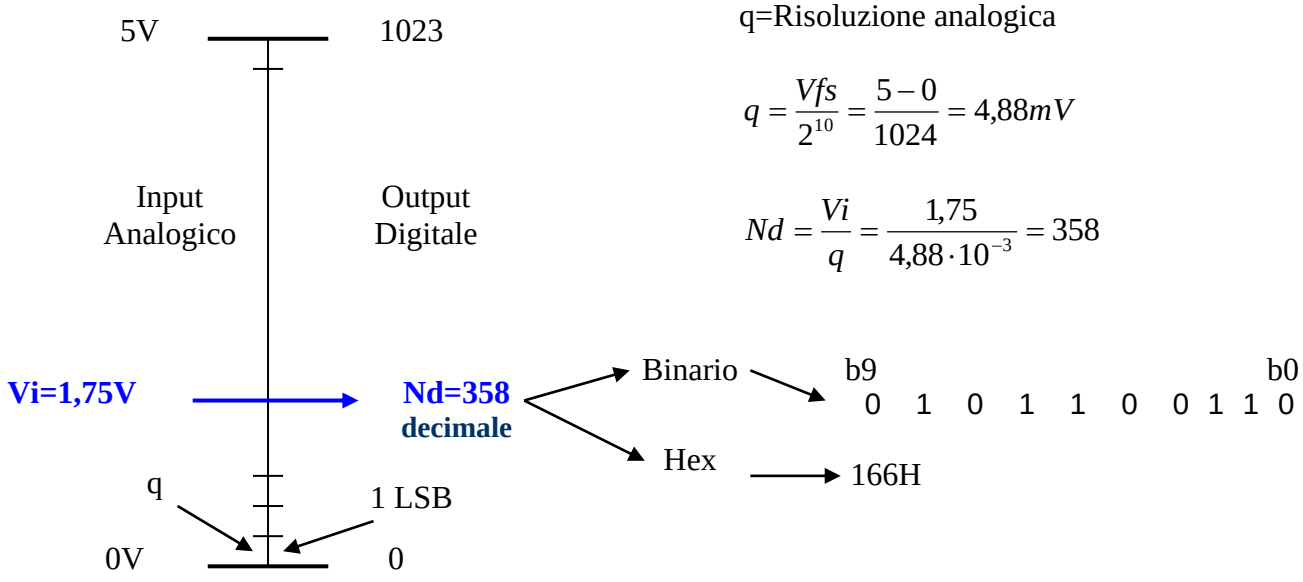


In tabella è riportata la corrispondenza canale-porta-pin

Canale	CB 220		CB 280	
	Porta	Pin	Porta	Pin
A/D Channel 0	P0	5	P24	37
A/D Channel 1	P1	6	P25	38
A/D Channel 2	P2	7	P26	39
A/D Channel 3	P3	8	P27	40
A/D Channel 4	P4	9	P28	56
A/D Channel 5	P5	10	P29	55
A/D Channel 6	P6	11	P30	54
A/D Channel 7	P7	12	P31	53



In figura è riportato lo schema a blocchi di un sistema in grado di acquisire, convertire e memorizzare 8 grandezze analogiche senza l'ausilio di un multiplexer esterno.
Se i segnali sono superiori ad 8 si deve inserire all'esterno del Cubloc un multiplexer, nelle prossime pagine verrà riportata la struttura di questo sistema.
In figura un esempio di conversione. Il valore analogico Vi=1,75V viene convertito nel numero binario 0101100110 corrispondente al numero decimale 358 oppure 166H.



q=Risoluzione analogica

$$q = \frac{Vfs}{2^{10}} = \frac{5-0}{1024} = 4{,}88mV$$

$$Nd = \frac{Vi}{q} = \frac{1{,}75}{4{,}88 \cdot 10^{-3}} = 358$$

Comandi Basic
Il convertitore A/D interno del Cubloc viene gestito tramite il comando ADIN oppure TADIN. Il comando TADIN opera con maggiore precisione (esegue 10 conversioni per lo stesso valore e restituisce la media). I due comandi hanno la stessa sintassi.

Sintassi di ADIN

Variabile=ADIN(Canale)

La variabile deve essere di tipo Integer, mentre il canale può andare da 0 a 7

Esempio 1

```
Const device = CB280
A Var integer
Input 24
A=ADIN(0)
```

Il programma legge il valore analogico presente sulla porta 24, corrispondente al canale 0 del A/D, lo converte e memorizza il risultato nella variabile A..

Esempio 2 – lettura, conversione e visualizzazione

```
Const device = CB280
Dim A As Integer
Dim B As Integer

Set display 2,0,0,128
Cls
Delay 200
Csroff

Input 24
Input 25

A= Adin (0)
B= Adin (1)

Locate 0,0
Print “CH0=” , Dec A
Locate 0,1
Print “CH1=” , Dec B
```

Il programma legge due valori analogici presenti sulle porte 24 e 25, li converte, li memorizza e li rappresenta in decimale sul display CLCD.

Esempio 3 – lettura, conversione e visualizzazione

Scrivere un programma Basic per CB280 in grado di:

1. leggere 4 valori analogici sulla porta P24 (Ch0) con un intervallo di 6 sec
2. memorizzare i valori in un Array
3. Calcolare il Totale e la Media
4. Visualizzare i Valori acquisiti, il Totale e la Media sul display CLCD

```
Const device = CB280

Dim A(4) As Integer
Dim x As Byte
Dim tot As Integer
Dim Media As Single

Set display 2,0,0,128
Cls
Delay 200
Csroff

Tot= 0
Input 24

For x= 0 To 3
A(x)= Adin (0)
tot= tot + A(x)
Delay 6000
Next

Media= tot/4

Locate 0,0
Print Dec A(0), “  “, Dec A(1), “  “, Dec A(2), “  “, Dec A(3)
Locate 0,1
Print “Tot= “, Dec Tot
Locate 0,2
Print “Media= “, Float Media
```

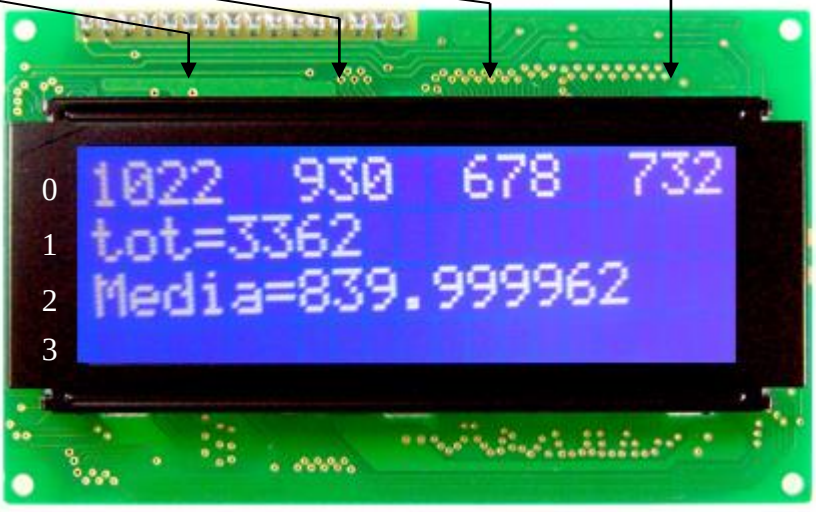
Variabili

A(4) | | | | |

x |

tot |

Media |



Esempio 4 – lettura, conversione, elaborazione e visualizzazione

Scrivere un programma Basic per CB280 in grado di:

1. leggere 100 valori analogici sulla porta P24 (Ch0) con un intervallo di 3 sec
2. memorizzare i valori in un Array e visualizzarli sul display
3. Determinare il valore massimo e quante volte si ripete tale valore all'interno dell'array
4. Visualizzare il valori sul display (Massimo e numero di ripetizione)

```
‘ ----------------------------------------------------------------
‘ --  lettura, conversione,  elaborazione e visualizzazione  --
‘ ----------------------------------------------------------------
Const Device = CB280
A       Var Integer (100)
MAX Var Integer
X       Var Byte
F       Var Integer

Set Display 2,0,0,50
Cls
Delay 300
Csroff

For X=0 To 99
A(X)=TADIN (0)
Locate 0,0
Print "X= ", Dec X," Valore= ", Dec4 A(X)
Delay 3000
Next

MAX=A(0)
For X=1 To 99
If A(X)>MAX Then
MAX=A(X)
End If
Next


F=0
For x =1 To 99
If A(X)=MAX Then F=F+1
Next

Locate 0,2
Print "Max=", Dec MAX, " F=", Dec F
```

Esempio 5 – Impostazione RTC (Scrittura e lettura orario)

Disegnare il circuito e scrivere un programma Basic per CB220 in grado di:

1. Tramite due pulsanti (P1 e P2) impostare l'orario nel formato HH:MM:SS
2. Il pulsante P1 per scegliere la parte da impostare (HH oppure MM oppure SS)
3. Il pulsante P2 per sceglie il valore
4. Tramite il pulsante P3, memorizzare i valori impostati all'interno del modulo RTC
5. Tutte le operazioni indicate devono essere visualizzate sul display CLCD

```
‘ ---------------------------------------------------------------
‘ --  Impostazione Orario sul Modulo RTC                    --
‘ ---------------------------------------------------------------
Const Device = CB220

Set Display 2, 0, 0, 128
Cls
Delay 300
Csroff

Dim conv As String *22
Dim data As String *8
Dim data_ita As String *8
Dim ora As String *8
Dim giorno As String *2
Dim tora As String
Dim x As Byte
Dim p As Byte
Dim xora(3) As Byte
Dim v(3) As String*2
xora(0)=0
xora(1)=0
xora(2)=0
p=0
v(0)="HH"
v(1)="MM"
v(2)="SS"
Opencom 1, 9600, 3, 24, 24
Locate 0,0
Print "Set Ora P0=Si P1=No"
Do
If Keyin (0,200)=1 Then
        imposta_ora
        scrivi_ora
Exit Do
Endif
  If Keyin (1,200)=1 Then Exit Do
Loop
```

```
‘ ------------------------------------------------------------------------
‘ --  Legge Orario sul RTC e lo visualizza sul display                  --
‘ ------------------------------------------------------------------------
Cls
Csroff
Do
        Putstr 1,"*RA"
                Do While Blen (1,1) >0
                Loop
        conv=Getstr(1,24)

        data=Mid(conv,2,8)    'data nel formato yy/mm/dd
        ora=Mid(conv,14,8)    'ora  nel formato hh/mm/ss
        giorno=Mid(conv,11,2) 'giorno della settimana
        data_ita=Right(data,2)+"/"+Mid(data,4,2)+"/"+Left(data,2)

        Locate 0,0
        Print data_ita, " ", ora, " ", giorno

        If ora>"12:06:00" Then High 4

        If ora>"12:07:00" Then Low 4
        Delay 1000
        Loop
End

‘ ---------------------------------------------------
‘ --  Subroutine Imposta Orario                    --
‘ ---------------------------------------------------
Sub imposta_ora ()
Cls
Csroff
Do
If Keyin (0,500)=1 Then Incr p
If p>2 Then p=0
If Keyin (1,500)=1 Then Incr xora(p)
If xora(0)>23 Then xora(0)=0
If xora(1)>59 Then xora(1)=0
If xora(2)>59 Then xora(2)=0
Locate 0,0
Print Dec2 xora(0),":",Dec2 xora(1),":",Dec2 xora(2),"    ", v(p)
Locate 0,1
If p=0 Then Print "^^       "
If p=1 Then Print "   ^^    "
If p=2 Then Print "      ^^ "
If Keyin(2,150) Then Exit Do
Loop
End Sub
```

```
‘ ------------------------------------------------------------------------
‘ --  Subroutine Memorizza Orario sul modulo RTC                        --
‘ ------------------------------------------------------------------------
Sub scrivi_ora ()
tora="*W2"+Dec2 xora(0)
Putstr 1,tora
Delay 250
tora="*W1"+Dec2 xora(1)
Putstr 1, tora
Delay 250
tora="*W0"+Dec2 xora(2)
Putstr 1, tora
Delay 250
End Sub
```