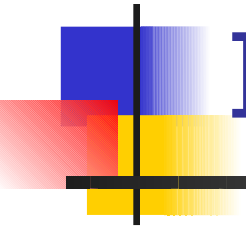


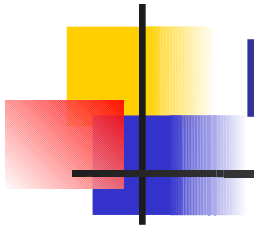
Istruzioni, algoritmi, linguaggi

Algoritmo per il calcolo delle radici reali di un'equazione di 2° grado

- Data l'equazione $ax^2+bx+c=0$, quali sono i valori di x per cui l'equazione è soddisfatta?

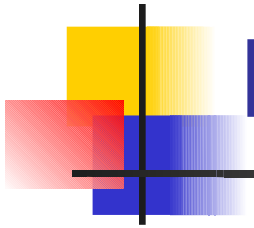
Algoritmo per il calcolo delle radici reali di un'equazione di 2° grado

- Algoritmo:
  - 1. Inizio dell'algoritmo.
  - 2. Acquisire i coefficienti a,b,c.
  - 3. Calcolare il valore $\Delta=b^2-4ac$.
  - 4. Se $\Delta<0$, allora non esistono radici reali; eseguire l'istruzione 8.
  - 5. Se $\Delta=0$, allora esistono due radici reali coincidenti, $x_1=x_2=-b/2a$; eseguire l'istruzione 7.
  - 6. Se $\Delta>0$, allora esistono due radici reali distinte, $x_1=(-b+\sqrt{\Delta})/2a$, $x_2=(-b-\sqrt{\Delta})/2a$; eseguire l'istruzione 7.
  - 7. Comunicare all'esterno i valori x_1 e x_2.
  - 8. Fine dell'algoritmo.



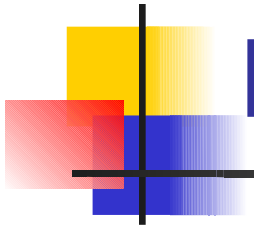
Proprietà degli algoritmi

- Perché una sequenza di istruzioni sia un algoritmo devono essere soddisfatti i seguenti requisiti:
  - finitezza;
  - generalità;
  - non ambiguità.



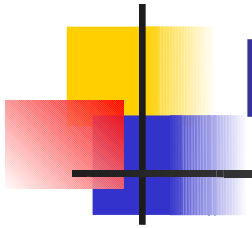
Proprietà degli algoritmi

- Finitezza:
  - il numero di istruzioni è finito;
  - ogni istruzione è eseguita in un intervallo finito di tempo;
  - ogni istruzione è eseguita un numero finito di volte.



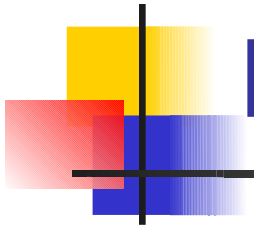
Proprietà degli algoritmi

- Generalità:
  - Un algoritmo fornisce la soluzione ad una classe di problemi, cioè:
    - dato un insieme di definizione, o *dominio*,
    - dato un insieme di arrivo, o *codominio,*
    - l'algoritmo può operare su tutti i dati appartenenti al dominio per fornire una soluzione all'interno del codominio.



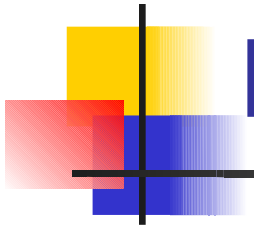
Proprietà degli algoritmi

- Non ambiguità:
  - le istruzioni sono definite in modo univoco;
  - non ci sono paradossi o contraddizioni;
  - il risultato dell'algoritmo è identico indipendentemente da chi lo sta eseguendo.



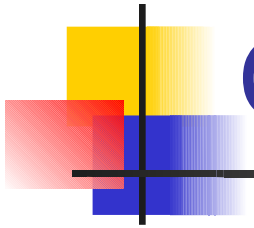
Esempio

- Testiamo l'algoritmo delle radici per le proprietà di algoritmo.
  - *Finitezza*: ci sono 8 istruzioni. Tutte sono eseguite al più una volta. Tutte impiegano un tempo finito per essere valutate o eseguite.
  - *Generalità*: in ingresso è ammissibile una qualsiasi terna di numeri reali e l'uscita è un numero reale.
  - *Non ambiguità*: le istruzioni sono ben definite: operazioni aritmetiche o confronti fra reali.



Descrizione degli algoritmi

- Le proposizioni usate da un linguaggio formale descrivono due entità:
  - le operazioni che devono essere eseguite (istruzioni);
  - gli oggetti (dati) sui quali si devono eseguire le operazioni.



Costanti e variabili

- I dati possono essere *costanti* o *variabili*.
- I dati costanti sono dati che rimangono inalterati durante l'esecuzione dell'algoritmo da quando ha inizio a quando termina.
- Esempio: nell'algoritmo del gioco dell'11 è una costante il numero di oggetti (2) presi dal primo giocatore.

Variabili

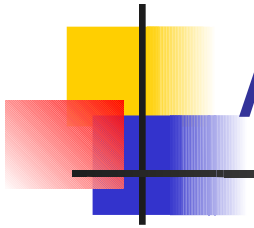
- I dati variabili o semplicemente *variabili* sono coppie

  $< nome , valore > .$

- Possono essere immaginati come scatole che hanno come etichetta il *nome* e come contenuto il *valore*.
- Alle variabili **deve** essere assegnato esplicitamente un valore.
- Al momento di inizio di un algoritmo le variabili hanno un valore **indeterminato**.
- Esempio: nell'algoritmo delle radici sono presenti le variabili a , b , c , x_1 e x_2.

Variabili

- Le variabili hanno lo stesso uso dei pronomi nel linguaggio naturale.
- Servono per riferirsi ad un oggetto (generalmente un valore numerico) indipendentemente dal valore specifico di questo.

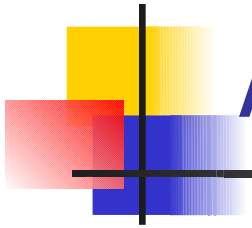


Assegnazione

- L'istruzione di assegnazione è quella particolare istruzione che permette di *definire* il valore attuale di una variabile.
- Il valore rimane *inalterato* fino ad una nuova assegnazione alla variabile.
- Forma generale:

  nome ← espressione

- L'assegnazione viene eseguita nei seguenti passi:
  - si valuta l'espressione di destra;
  - si attribuisce il valore determinato alla variabile.



Assegnazione

- **Regola**
  - Ogni volta che una variabile appare a destra dell'istruzione di assegnazione $\leftarrow$, è necessario che un valore sia già stato assegnato a quella variabile.
- Esempio:
  - nel caso $a \leftarrow 2$, $b \leftarrow 3$, $c \leftarrow a+b$, allora si ha $c=5$.
  - nel caso $x \leftarrow 2$, $x \leftarrow x+3$, allora si ha $x=5$.

Vettori

- Le variabili considerate finora sono dette *variabili scalari*.
- Una *variabile vettore* (array) è una coppia

  < nome , insieme di valori >

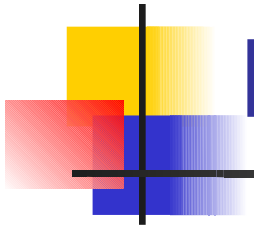
- Può essere immaginata come una scatola che ha un nome e che è divisa in tanti comparti, ognuno dei quali è numerato e può contenere un valore.

Vettori

- Ogni valore è individuato dal nome della variabile seguito dal numero del comparto detto *indice*.
- La notazione usata è

$$A[i]$$

- La *dimensione* di un vettore è il numero dei suoi elementi.

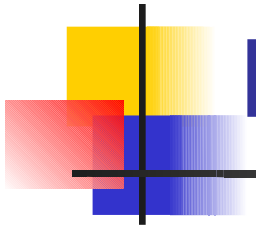


Dimensione dei vettori

- Dato un vettore di dimensione *N*, gli indici vanno per convenzione o da 0 a *N*-1 oppure da 1 a *N*.
- NOTA: in fase di dichiarazione di una variabile vettore si specifica la sua dimensione che non è più modificabile successivamente.

Assegnazione di un valore a un vettore

- L'istruzione di assegnazione ad un vettore è analoga a quella per una variabile ordinaria.
- La differenza sta nel fatto che si deve specificare anche a quale posizione ci stiamo riferendo.
- Esempio:
  - supponiamo di avere un vettore *A* di dimensione 10;
  - allora possiamo scrivere $A[3] \leftarrow 2$ oppure $A[0] \leftarrow 5$;
  - ma non possiamo scrivere $A[-5] \leftarrow 2$ oppure $A[122] \leftarrow 5$, perché nel primo caso non si possono contare posizioni negative, e nel secondo non si possono indicare posizioni successive a 9 (o 10).

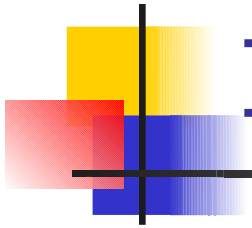


Matrice

- È l'estensione del concetto di vettore.
- Una matrice è un insieme di valori che sono indicizzati facendo ricorso a due o più indici.
- La notazione usata è

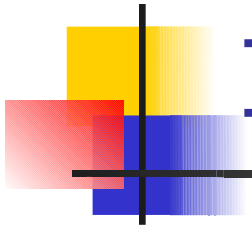
$$M[i,j]$$

- Per una matrice a due dimensioni l'indice *i* è detto indice *riga* e *j* indice *colonna*.
- Per una matrice l'assegnazione avviene come per un vettore, con la differenza di dover specificare due indici.
  - Possiamo scrivere $A[3,2] \leftarrow 2$ oppure $A[0,0] \leftarrow 5$.



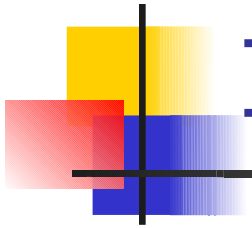
Istruzioni

- Ma quali sono le istruzioni possibili in un calcolatore moderno?
- Cosa riesce a fare direttamente?
- Quali sono i blocchi elementari che possiamo comporre per costruire espressioni sempre più complesse, programmi sempre più sofisticati?



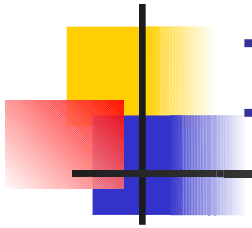
Istruzioni

- Le istruzioni possono essere suddivise in 5 categorie:
  - *istruzioni operative;*
  - *istruzioni di salto;*
  - *istruzioni di inizio/fine esecuzione;*
  - *istruzioni di ingresso/uscita;*
  - *istruzioni condizionali, o di controllo.*



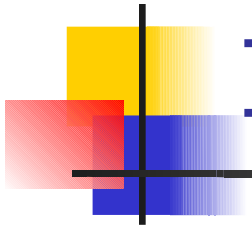
Istruzioni operative

- Istruzioni che producono un risultato se eseguite.
- Ne fanno parte le operazioni aritmetiche e le assegnazioni.
- Esempi:
  - l'istruzione: 5+3;
  - l'istruzione: $x \leftarrow 2$;
  - l'istruzione: 3 *mod* 2 (*mod* sta per modulo, ovvero resto della divisione intera).



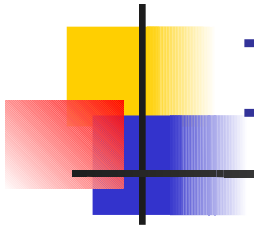
Istruzioni di salto

- Istruzioni che alterano il normale ordine di esecuzione delle istruzioni di un algoritmo, specificando esplicitamente quale sia la successiva istruzione da eseguire.
- Si distinguono in istruzioni di salto condizionato e incondizionato, a seconda che debba essere verificata una condizione per poter attuare il salto oppure no.
- Nei linguaggi di programmazione "strutturati", come diremo tra poco, non sono usate.



Istruzioni di inizio/fine

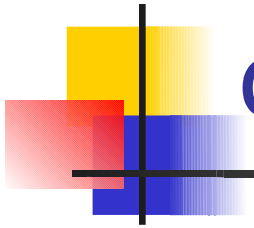
- Indicano quale istruzione dell'algoritmo debba essere eseguita inizialmente e quale determini la fine dell'esecuzione.



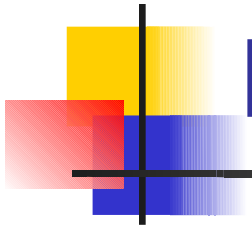
Istruzioni di ingresso/uscita

- Istruzioni che indicano una trasmissione di dati o messaggi fra l'algoritmo e tutto ciò che è esterno all'algoritmo.
- Tali istruzioni si dicono
  - di ingresso o *lettura* quando l'algoritmo riceve dati dall'esterno;
  - di uscita o *scrittura* quando i dati sono comunicati dall'algoritmo all'esterno.

Istruzioni condizionali, o di controllo



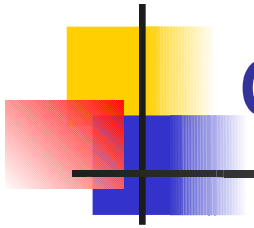
- Istruzioni che controllano il verificarsi di condizioni specificate e che in base al risultato determinano quale istruzione eseguire.
- Nota: si altera il flusso del programma in funzione della condizione (se è vera o falsa).
- Esempio: se … allora … altrimenti (*if* … *then* … *else*).



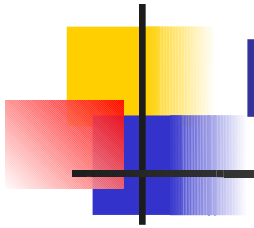
Esempio

- Nell'algoritmo delle radici
  - le istruzioni 1,8 sono di inizio/fine;
  - le istruzioni 2,7 sono di ingresso/uscita;
  - l'istruzione 3 è operativa;
  - (parte dell') istruzione 4 è di salto;
  - le istruzioni 5,6 sono condizionali.

I predicati nelle istruzioni di controllo

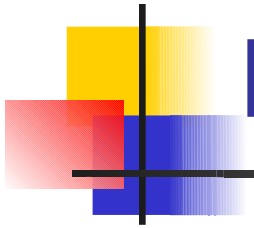


- Le istruzioni di controllo devono determinare (calcolare) la verità o falsità di un enunciato.
- Tale enunciato viene chiamato proposizione e si parla di valore di verità di tale proposizione.



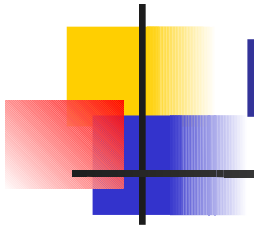
Proposizioni

- Una *proposizione* è un costrutto linguistico del quale si può dire con certezza se è vero o falso.
- Il *valore di verità* di una proposizione è l'essere vera o falsa di tale proposizione.
- Esempi:
  - "3 è un numero pari" è una proposizione;
  - "incrementa *x* di 10" non è una proposizione.



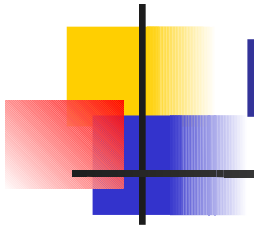
Predicati

- È un *predicato* una proposizione che contiene delle variabili e in cui il valore delle variabili determina il valore di verità del predicato.
- Esempio: la variabile *età* è minore di 30.



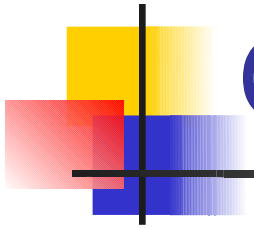
Predicati

- La composizione di un predicato avviene tramite i seguenti *operatori relazionali* :
  - =: uguale, ≠: diverso;
  - >: maggiore, ≥: maggiore o uguale;
  - <: minore, ≤: minore o uguale.



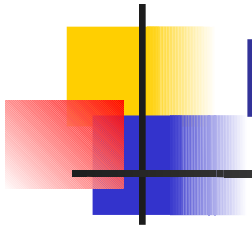
Predicati semplici e composti

- Un predicato che contiene un solo operatore relazionale è detto predicato semplice.
- Gli operatori relazionali possono essere combinati con i seguenti *operatori logici* o *connettivi* : *NOT, AND, OR*.
- Un predicato che contiene più operatori relazionali combinati tramite uno o più operatori logici è detto composto.



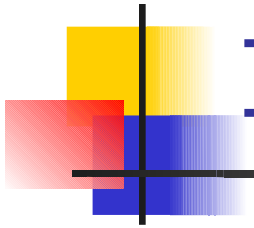
Operatori logici

- NOT:
  - dato un predicato *p*, *NOT p* è vero quando *p* è falso e viceversa;
- AND:
  - dati due predicati *p* e *q*, *p AND q* è vero solo quando entrambi *p* e *q* sono veri e falso altrimenti;
- OR:
  - dati due predicati *p* e *q*, *p OR q* è falso solo quando entrambi *p* e *q* sono falsi e vero altrimenti;



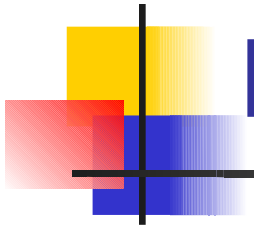
Esempio

- Utilizzate un opportuno predicato per esprimere ciascuno dei seguenti concetti:
  - età compresa fra 18 e 60 anni;
  - altezza superiore a 1,90m oppure peso superiore a 100Kg;
  - anno multiplo di 4 ma non multiplo di 100.



Istruzioni e calcolatore

- Per rendere possibile l'esecuzione di un programma da parte di un calcolatore, è necessario che le istruzioni siano:
  - interpretabili;
  - attuabili.
- Perché le istruzioni siano interpretabili devono essere codificate in binario.
- Perché le istruzioni siano attuabili deve esistere un dispositivo nel computer che materialmente esegue l'istruzione (come vedremo):
  - unità centrale;
  - unità logico-aritmetica.



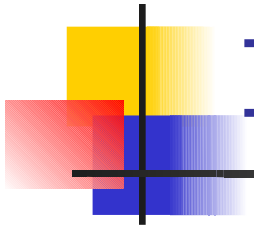
Linguaggio macchina

- L'insieme di istruzioni direttamente interpretabili ed eseguibili da parte di un calcolatore si chiama linguaggio macchina.
- Il linguaggio macchina è detto di *basso livello* perché dipende dalle specifiche caratteristiche del calcolatore:
  - il tipo di istruzioni elementari disponibili varia;
  - il modo di specificare gli indirizzi dei dati varia.

Linguaggio ASSEMBLER

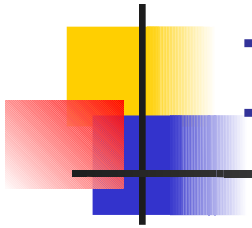
- Scrivere le istruzioni in codifica binaria è un compito difficile per gli esseri umani.
- Per questo si introduce il linguaggio assembler che altro non è che una traduzione simbolica delle varie istruzioni.

Codice	Istruzione	Descrizione
1011011	ADD	somma
10001110	JMP	salto
1000110	LDA	load
10001110	STA	store



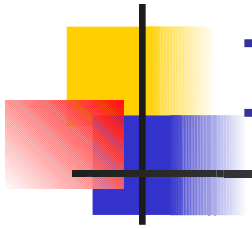
Istruzioni macchina

- Le istruzioni in linguaggio macchina sono composte da un numero variabile di parti (da 1 a 4).
- La prima parte costituisce il codice operativo ed identifica il tipo di istruzione.
- Esempio:
  - somma;
  - AND logico;
  - memorizzazione;
  - prelievo da memoria;
  - salto;
  - ecc...



Istruzioni macchina

- Le restanti parti specificano i dati su cui deve operare l'istruzione.
- Esempio:
  - una somma ha bisogno di 3 parametri per specificare le celle di memoria in cui sono memorizzati i due addendi e la cella di memoria in cui memorizzare il risultato;
  - un'istruzione di salto ha bisogno di un parametro per specificare l'indirizzo di memoria in cui è contenuta l'istruzione successiva da eseguire.

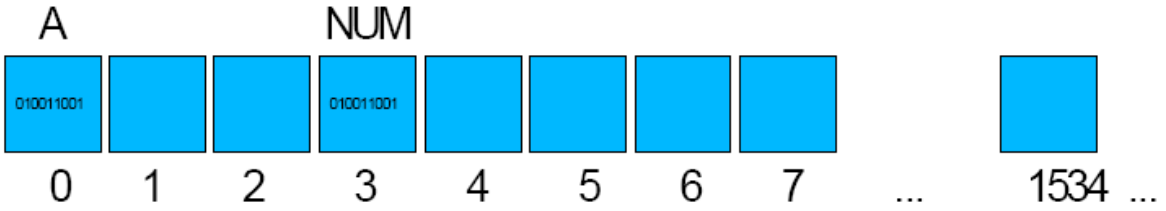


Istruzioni macchina

- Per specificare un dato si usa il concetto di indirizzo.
- Esistono una serie di celle di memoria che sono identificabili con un numero progressivo.
  - Cella 0, cella 1, cella 2, ecc... .
- Tale numero è l'indirizzo della cella.
- All'interno di una cella si memorizza di volta in volta un dato numerico diverso.
- In assembler si può utilizzare un'etichetta simbolica per indicare un indirizzo.

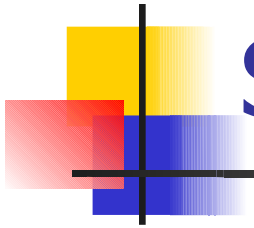
La memoria

- La memoria di un calcolatore si può vedere come un insieme di "celle".
- Ogni cella può contenere un dato.
- Un dato non è altro che un insieme di bit (8,16,32 o 64).
- È possibile riferirsi ad un cella indicandone il numero progressivo …
- … oppure tramite un nome simbolico: il nome di una variabile!



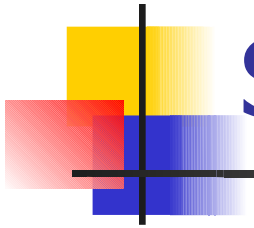
Esempio di programma in assembler

```
        BEGIN           ;inizio programma
START:  IN      1       ;lettura dato da unità 1
        JPZ     FINE    ;controllo se il dato letto è 0
        ADD     RIS     ;somma il dato a RIS
        STA     RIS     ;memorizza la somma in RIS
        JMP     START   ;torna a leggere un nuovo dato
FINE:   LDA     RIS     ;il ciclo di lettura e somma è finito
        OUT     2       ;scrittura del risultato sull'unità 2
        HLT             ;fine elaborazione
RIS:    WRD     0       ;per mantenere somma parziale
        END     START   ;fine programma
```



Spiegazione

- IMPORTANTE: esiste una variabile (registro accumulatore) sottointesa, in cui si memorizza o recupera un eventuale dato.
- IN: si legge da un'unità specificata (la 1 potrebbe essere la tastiera, ad esempio) un dato e si pone nel *registro accumulatore.*
- JPZ: esegue un salto condizionato al fatto che nel *registro* vi sia un valore nullo; questa sarà la condizione per terminare il ciclo di esecuzione.
- ADD: somma quanto contenuto nel *registro* al dato indirizzato (RIS) e pone il risultato nel registro stesso.



Spiegazione

- STA (STORE A): memorizza quanto contenuto nel *registro* nell'indirizzo indicato (nell'esempio è RIS).
- JMP: esegue un salto incondizionato a un dato indirizzo (nell'esempio è START).
- LDA (LOAD A): recupera da un indirizzo (nell'esempio è RIS) e pone il risultato nel *registro*.
- OUT: scrive su un'unità (nell'esempio è l'unità 2) il dato presente nel *registro*.
- HLT: interrompe l'esecuzione.
- WRD: riserva una cella di memoria per immagazzinare un dato.