# Calcolatori Elettronici
# Le istruzioni macchina

Ing. Gestionale e delle Telecomunicazioni

A.A. 2009/10

Gabriele Cecchetti

## Le istruzioni macchina

- Sommario:
  - Generalità
  - Tipi di linguaggio macchina
  - Tipi di architettura
  - Formalismo RTL
  - Esempi semplici di istruzioni
  - Classi di istruzione
  - Modi di indirizzamento
  - Bit di esito
  - Alcuni esempi di programma
  - Ingresso e uscita
  - Struttura ed uso dello Stack
  - Alcune istruzioni macchina
- Riferimenti
  - C. Hamacher, “Introduzione all’architettura del Calcolatore”, cap. 4.
  - G. Bucci “Architetture e organizzazione dei Calcolatori Elettronici – Fondamenti”, Cap. 5

Alcune nozioni fondamentali

# GENERALITÀ



## Istruzione Macchina (1/4)

- L'istruzione macchina è il compito elementare eseguibile da parte del processore.
- Essa si distingue in forma simbolica e numerica:
  - simbolica: come viene scritta da parte del programmatore, in modo leggibile e facilmente interpretabile
  - numerica (o binaria): come è rappresentata in memoria nel programma in esecuzione, in forma adatta al processore

## Istruzione Macchina (2/4)

- **Ogni processore ha uno specifico insieme (o repertorio) di istruzioni macchina**:
  - *il linguaggio macchina*
- Le istruzioni macchina di processori diversi differiscono, tanto o poco.
- Ma ci sono sempre svariati concetti e caratteristiche comuni a tutti, o quasi, i processori.



## Istruzione Macchina (3/4)

- **Il linguaggio macchina in forma simbolica è chiamato anche linguaggio assemblatore o *assembly language***.
- L'assemblatore (*assembler*) è uno strumento SW che **esamina il programma in linguaggio macchina simbolico**.
- Se non ha errori, **lo traduce** (o assembla) **generandone la forma numerica corrispondente**.

## Istruzione Macchina (4/4)

- Di seguito si danno le caratteristiche fondamentali che si trovano nelle istruzioni di ogni linguaggio macchina, per un tipo di processore classico con registri interni.

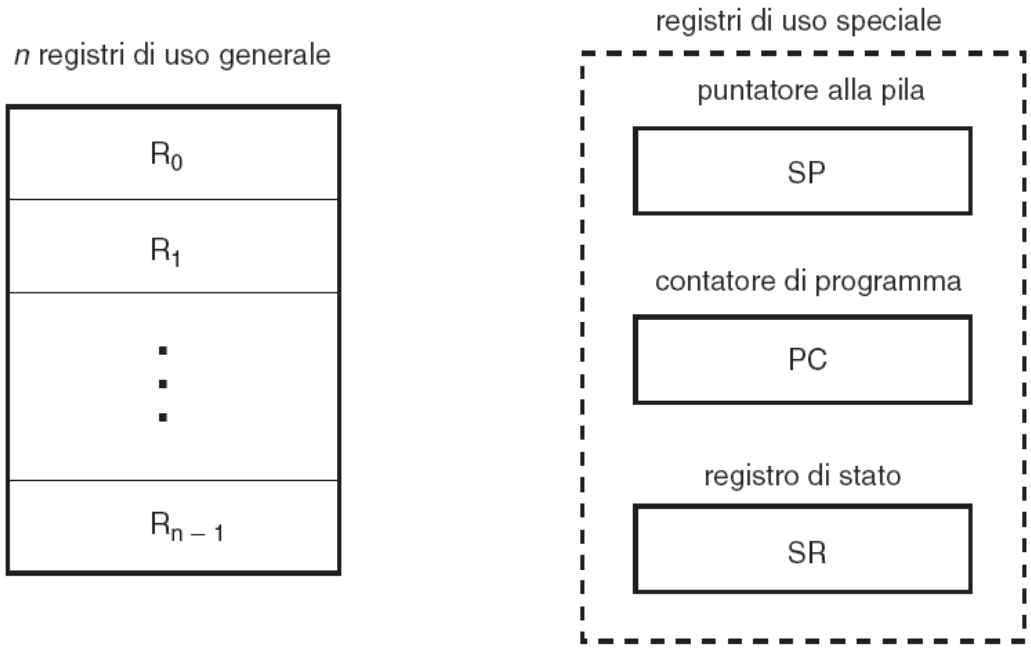
- Più avanti si mostrerà un linguaggio macchina specifico della famiglia di processori Intel x86
- Molte delle caratteristiche fondamentali si ritrovano, cambia solo la notazione.



## Modello di Processore

- La maggior parte dei processori è costituita da *alcuni registri interni* e *un'unità aritmetica-logica*:
  - **banco di registri di uso generale**: servono per i dati
  - **registri di uso speciale**: servono per il controllo
- I registri interni contengono i dati da elaborare correntemente e varie informazioni di controllo necessarie per eseguire il programma.
- Molti dei registri interni del processore registri sono visibili da parte del programmatore in linguaggio macchina, e si possono nominare nelle istruzioni macchina stesse.

# Modello di Processore




Modello generico di processore, vale per qualunque processore ragionevole.



# Registri di Uso Generale

- I registri di uso generale (*general purpose*) hanno vari nomi, secondo la marca di processore, ma quasi sempre hanno anche nome numerato: da $R_0$ a $R_{n-1}$ (ci sono esattamente $n \geq 1$ registri, $n = 8, 16, 32, \ldots$).
- I registri di uso generale:
  - servono soprattutto per contenere i dati da elaborare (talvolta indirizzi – *sono anch'essi numeri interi*).
  - sono per lo più equivalenti tra sé, e dunque interscambiabili, salvo rari casi da vedere in specifico. Talvolta si possono usare in gruppo, come un blocco unico.

## Registri di Uso Speciale

- I registri di uso speciale (*special purpose*) si usano individualmente, per funzioni specifiche, e hanno nome proprio.
- Talvolta il registro speciale esercita la propria funzione in modo del tutto automatico e non modificabile, come per esempio il contatore di programma.
- Oppure la esercita in modo soggetto a restrizioni di vario tipo, secondo la funzione del registro.



## Registri di Uso Speciale

- *Contatore di programma* o **PC** (*program counter*):
  - contiene l'indirizzo dell'istruzione macchina da prelevare ed eseguire
- *Puntatore alla pila* o **SP** (*stack pointer*):
  - contiene l'indirizzo della cima della pila (*stack*), e si usa per la gestione del sottoprogramma (routine)
- *Registro di stato* o **SR** (*status register*, talvolta indicato come **PSW**, *program status word*):
  - contiene vari bit che indicano lo stato del processore (modo utente-sistema, bit di esito, controllo di interruzione, e simili)

## Funzionamento del Processore

- Le istruzioni macchina e i dati da elaborare si trovano in memoria centrale.
- Da qui, l'istruzione macchina da eseguire correntemente e i dati che essa deve elaborare vengono letti da parte del processore e scritti nei registri interni (cioè vengono caricati).
- Al termine i risultati dal calcolo vengono (eventualmente) riscritti in memoria centrale (cioè vengono memorizzati).



## La CPU

- Legge le istruzioni dalla memoria,
- le decodifica, e
- genera i comandi che fanno eseguire le azioni previste da ogni specifica istruzione.

## Ciclo di Prelievo-Esecuzione (1/2)

1) Il processore preleva (***fetch***) da memoria centrale l'istruzione macchina da eseguire correntemente.
2) Il processore decodifica (***decode***) l'istruzione, cioè la analizza e ne determina il significato.
3) Il processore esegue (***execute***) l'istruzione, elaborando i dati opportuni e producendo il risultato:
   - i dati, si trovano già nei registri *oppure* sono caricati dalla memoria,
   - il risultato, viene lasciato in un registro *oppure* viene memorizzato.



## Ciclo di Prelievo-Esecuzione (2/2)

4) Allo stesso tempo il processore decide quale sarà la prossima istruzione da eseguire:
   - spesso è quella in memoria consecutiva all'istruzione corrente
   - talvolta è collocata altrove in memoria e va reperita (salto)
5) Poi il ciclo si ripete, ma con la nuova istruzione.
   → *il processore è essenzialmente una macchina ciclica*.

# La memoria: le operazioni fondamentali

- **Lettura**:
  - fornito l'indirizzo della cella da leggere si trasmette alla memoria il relativo comando e la memoria risponde con il contenuto.
- **Scrittura**:
  - fornito il dato e l'indirizzo della cella da scrivere si trasmette alla memoria il relativo comando.

- Queste operazioni possono comportare la lettura/scrittura di *byte* - *half-word* - *word*
  - leggendo/scrivendo più bit aumenta il parallelismo e quindi l'efficienza di queste operazioni.



# Struttura della Memoria (1/2)

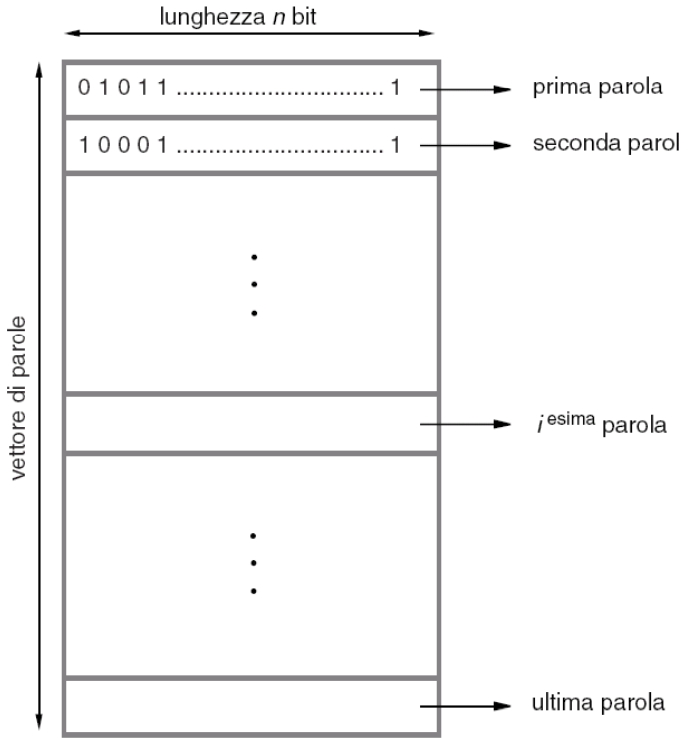
- Fondamentalmente la memoria del calcolatore è costituita da una sequenza (vettore) di parole (o celle, o locazioni) binarie.
- Ogni parola è una stringa (successione) di un numero fissato di bit:
  - 8 bit: carattere o *byte*
  - 16 bit: parola (ordinaria) (*word*)
  - 32 bit: parola doppia (*double* o *long word*)
  - 64 bit: parola quadrupla (*quad word*)
- Ogni parola ha un contenuto o valore, che è la sequenza di bit memorizzata, e un indirizzo (*address*), che serve per identificare la parola.

# Struttura della Memoria (2/2)

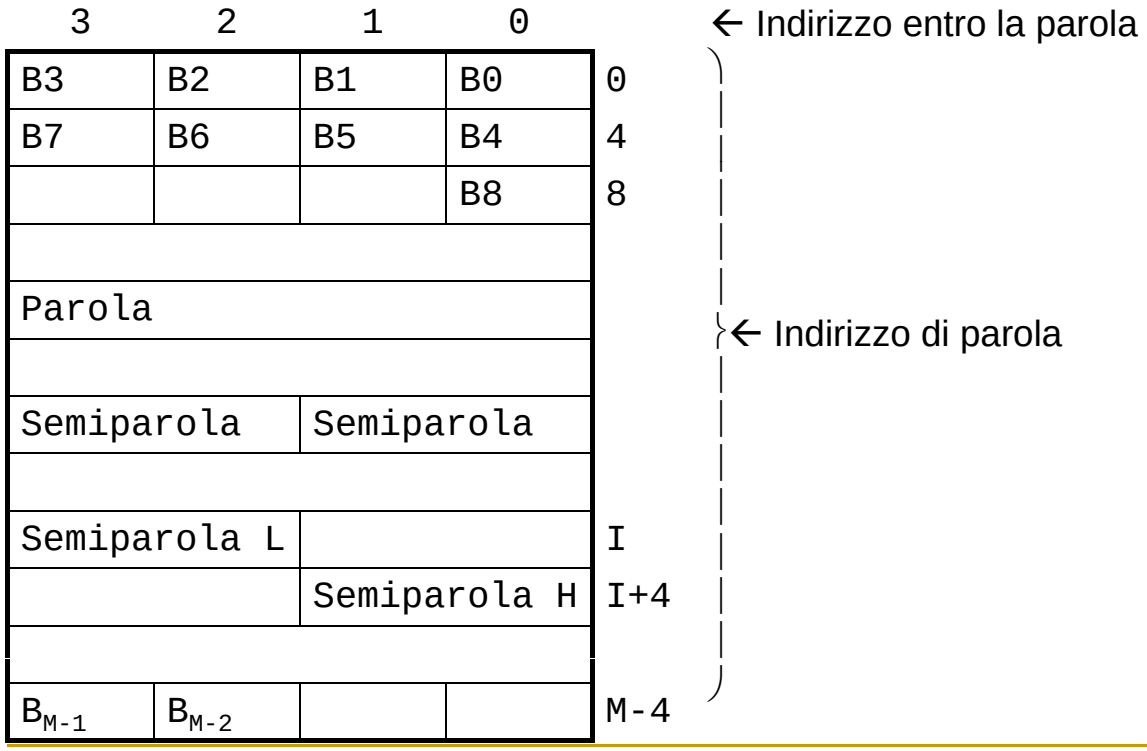
- L'indirizzo è un numero progressivo
  - da 0 ad a M-1, dove M è la dimensione della memoria, e
  - indica la posizione della parola all'interno della sequenza (vettore) di parole costituenti l'intera memoria centrale del calcolatore.
- La "parola" di memoria, a seconda del contesto, può indicare due entità diverse:
  - il contenuto, cioè il valore corrente della parola
  - oppure il contenitore del valore, e non il valore che correntemente c'è dentro
- Il contesto aiuta a capire: se serve per non cadere in ambiguità, per intendere la parola come contenitore si dica cella o locazione.



# Struttura della Memoria



# Memoria a 32 bit di M byte

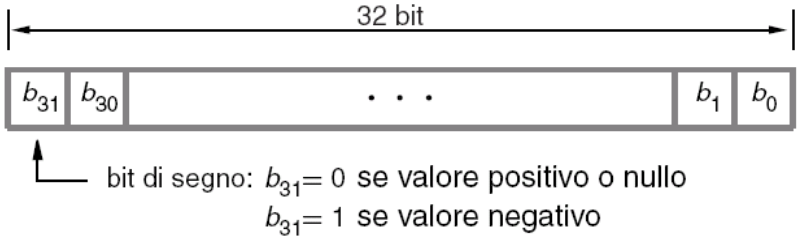




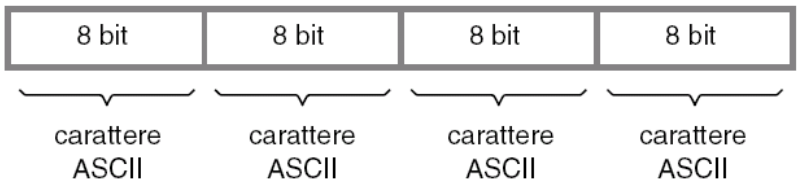

# Dato in Memoria

- La memoria contiene istruzioni macchina (programma) e dati (da elaborare).
- Se il tipo di dato è elementare, il dato occupa una sola parola o parte di essa:
  - **valore logico**: un bit (o un'intera parola se il valore logico è assimilato a un numero intero, come in C)
  - **carattere**: generalmente un solo byte (codice ASCII)
  - **numero intero** (naturale o complemento a 2): una parola o una parola doppia (secondo i casi)
- Se il tipo di dato è complesso (numero reale) o strutturato (vettore, record, ecc), il dato occupa una successione di parole contigue.

# Dato in Memoria




(a) parola di memoria da 32 bit contenente un numero intero relativo rappresentato in complemento a due con $n = 32$ bit




(b) parola di memoria da 32 bit contenente quattro caratteri ASCII



# Indirizzamento di Memoria

- In genere l'elemento di memoria minimo che il processore può leggere (caricare) o scrivere (memorizzare) è il carattere, cioè un byte (8 bit).
- Pertanto, quasi sempre l'indirizzo di memoria si riferisce al byte e ogni byte è numerato progressivamente, da 0 in poi.
- Leggendo o scrivendo due byte consecutivi, il processore lavora con una parola, quattro byte una parola doppia, ecc.
- Per ragioni tecnologiche, di norma il numero di byte consecutivi costituenti una parola è una potenza di due: 2, 4, 8 ed eventualmente di più.
- Il processore non può operare simultaneamente su byte non consecutivi in memoria.

## Indirizzamento di Memoria

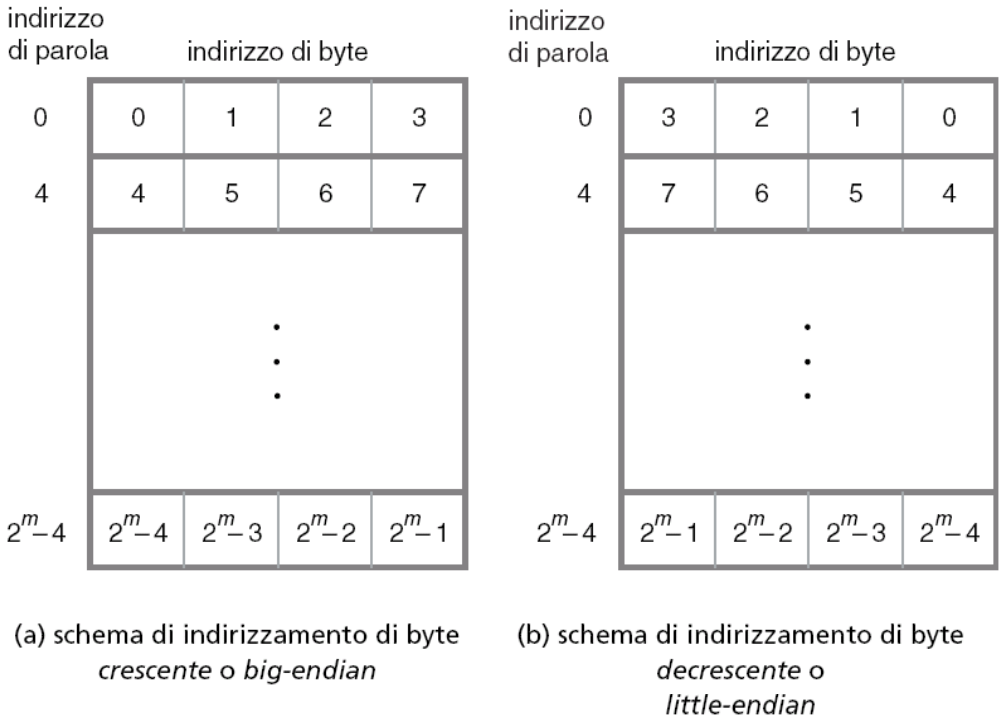
- Spesso si fanno coesistere due o più schemi di indirizzamento di memoria distinti, ma ben armonizzati:
  - uno schema per i byte (quasi sempre presente)
  - uno schema per la parola (ordinaria)
  - uno schema per la parola doppia
  - eventualmente altri …
- Allineamento degli schemi: l'indirizzo è multiplo della lunghezza della parola, misurata in byte.
- Esempio: la parola ordinaria (due byte) può avere solo indirizzo di valore pari: 0, 2, 4, ecc.



## Ordinamento di Byte

- All'interno di una parola, i byte componenti possono essere ordinati per indirizzo:
  - crescente (metodo big-endian)
  - decrescente (metodo little-endian)
- Processori diversi usano l'uno o l'altro metodo, ma non c'è differenza reale.
- Talvolta si può configurare il processore per usare uno dei due metodi a scelta.

# Ordinamento di Byte




(a) schema di indirizzamento di byte *crescente* o *big-endian*

(b) schema di indirizzamento di byte *decrescente* o *little-endian*



# Indirizzamento di Memoria

- Qualche processore coordina in modo non allineato gli schemi di indirizzamento di parola:
  - l'indirizzo di parola non è soggetto a restrizione, può assumere qualunque valore indipendentemente dal numero di byte componenti la parola stessa

  →*È più complesso da gestire e pertanto rallenta l'operazione di lettura e scrittura di memoria.*

  →Il disallineamento è una funzione poco frequente e usata raramente, e spesso, anche se il processore ne dispone, viene tenuta disabilitata.

## Istruzione in Memoria

- L'istruzione macchina, in forma numerica cioè eseguibile da parte del processore, è anch'essa contenuta in una o più parole di memoria consecutive, come il dato.
- Il numero di parole consecutive occupate da un'istruzione macchina dipende:
  - da come l'istruzione è codificata in bit
  - da quanto l'istruzione è complessa e lunga
- Spesso l'istruzione occupa una sola parola, in generale, dipende dal processore.



## Il Seguito …

- Qui sono presentate le nozioni fondamentali sulle istruzioni macchina.
- Non si fa riferimento a un linguaggio macchina di un processore (o famiglia) specifico.
- Le istruzioni sono citate e spiegate con nomi "generici" e intuitivi, facili da ricordare.
- Tuttavia, parecchi di tali nomi si ritrovano nei linguaggi macchina effettivi, con poche variazioni lessicali o precisazioni, in particolare nei linguaggi macchina di Motorola a Intel.

La struttura generale delle istruzioni macchina

# IL LINGUAGGIO MACCHINA



## La codifica delle istruzioni

- Esistono diverse architetture di calcolatori ognuna delle quali può prevedere modelli diversi di istruzioni e dati.
- Per semplicità, supponiamo di avere delle istruzioni che hanno sempre lunghezza pari a 32 bit.
- La CPU a cui faremo riferimento può disporre di diversi registri a 32 bit e porzioni di essi a 16 e a 8 bit.

## Forma Numerica

- L'istruzione macchina simbolica viene codificata in forma numerica da parte dell'assemblatore.
- La forma numerica è quella riconoscibile da parte del processore, e da esso direttamente eseguibile.
- In forma numerica gli elementi simbolici costituenti l'istruzione (nome, argomenti) vengono codificati in numeri binari: parole intere o parti di parola;
  - tali elementi sono chiamati *campi* (dell'istruzione).



## Elementi Simbolici

- Gli elementi simbolici dell'istruzione macchina, che vengono codificati numericamente, sono tipicamente:
  - **nome dell'istruzione** (o codice mnemonico) ⇒ campo *codice operativo*
  - **nome di registro** ⇒ campo *indirizzo di registro*
  - **costante simbolica** ⇒ campo *numero binario*
  - **etichetta** ⇒ campo *indirizzo in forma binaria*
- Una volta convertiti in forma numerica (ciascuno in una stringa di bit, più o meno lunga) vengono concatenati a costituire l'istruzione completa.

## Formato di Istruzione

- La forma numerica dell'istruzione macchina dipende molto dal tipo di processore dove essa dovrà essere eseguita.
- Non esiste una forma numerica universale adatta per tutti i processori.
- Produttori diversi hanno ciascuno le proprie convenzioni, valide per i rispettivi processori.
- Per esempio, le forme numeriche dei processori di famiglia Motorola e Intel sono diversissime (*le forme simboliche, pur differenti anch'esse, hanno invece molti aspetti comuni*).



## Formato di Istruzione

- L'istruzione macchina in forma numerica occupa una o più parole di memoria, secondo la dimensione della parola e la complessità dell'istruzione stessa (cioè il numero di elementi simbolici costituenti l'istruzione, e pertanto il numero di campi numerici corrispondenti).
  - Parola corta (8 o 16 bit):
    - in genere, l'istruzione ingombra due o più parole consecutive.
  - Parola lunga (32 o 64 bit):
    - In genere, l'istruzione in genere ingombra esattamente una parola.

# Formato di Istruzione

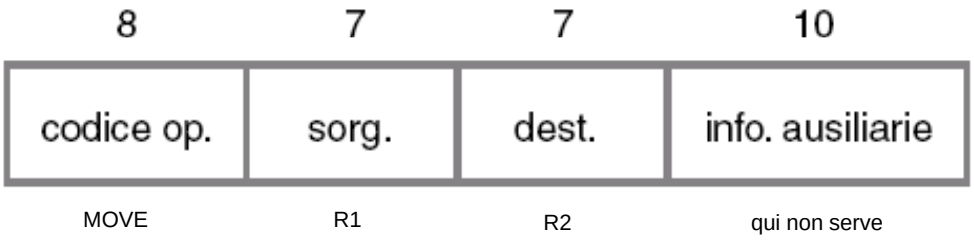
- L'istruzione macchina in forma numerica è suddivisa in campi (stringhe di bit) concatenati.
- Ogni campo codifica una certa informazione (nome, registro, indirizzo, ecc).
- La prima parola (talvolta l'unica) dell'istruzione, si chiama parola di codice operativo.
- Il campo codice operativo, che esprime il nome, è sempre presente, ed è generalmente il primo.
- Tutti gli altri campi sono più o meno facoltativi, dipende dall'istruzione che si vuole codificare.
- Le eventuali parole consecutive (di solito codificano un indirizzo o una costante), si chiamano parole aggiuntive.



# Formato di Istruzione

- I campi di un'istruzione macchina NON si possano combinare in modo libero.
- Esistono delle regole, e di solito il linguaggio ammette tre o quattro schemi di combinazione di campi, tramite cui si possono codificare numericamente tutte le istruzioni macchina simboliche ammissibili.
- Tali regole costituiscono propriamente il formato numerico del linguaggio, e variano secondo la famiglia di processori considerata.
- Nel seguito ci si limita a dare qualche esempio - per i dettagli si rimanda al manuale del linguaggio macchina di interesse.

# MOVE R1, R2




(a) codifica binaria a una parola

la parola di memoria è da 32 bit (4 byte)

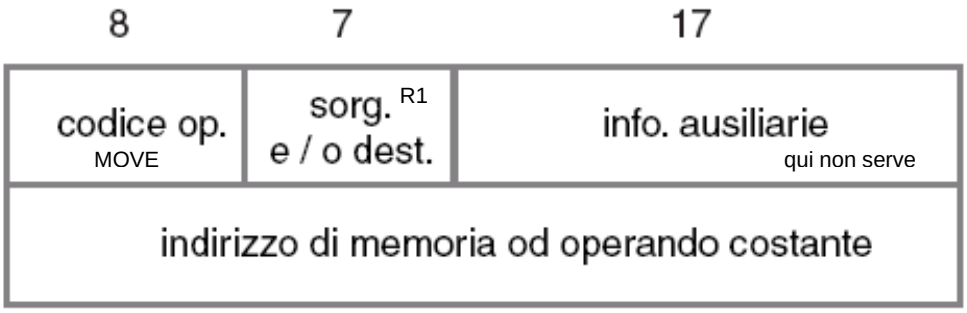
l'istruzione ingombra una parola

7 bit per l'indirizzo di registro e il modo di indirizzamento associato

formato adatto se l'istruzione è a due arg. e non nomina indirizzi



# MOVE R1, indirizzo




(b) codifica binaria a due parole

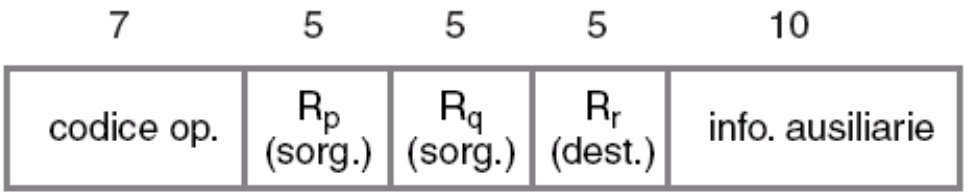
la parola di memoria è da 32 bit (4 byte)

l'istruzione ingombra due parole - la seconda contiene l'indirizzo

7 bit per l'indirizzo di registro e il modo di indirizzamento associato

formato adatto se l'istruzione è a due arg. e nomina un indirizzo

# ADD R1, R2, R3




(d) codifica binaria a una parola per formato di istruzione ternaria

la parola di memoria è da 32 bit (4 byte)

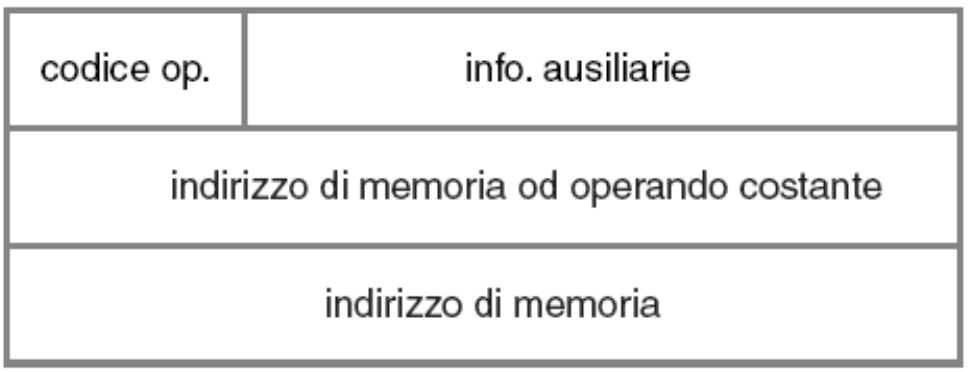
l'istruzione ingombra una parola - la seconda contiene l'indirizzo

5 bit per l'indirizzo di registro e il modo di indirizzamento associato

formato adatto se l'istruzione è a tre arg. e non nomina indirizzi



# MOVE indirizzo1, indirizzo2

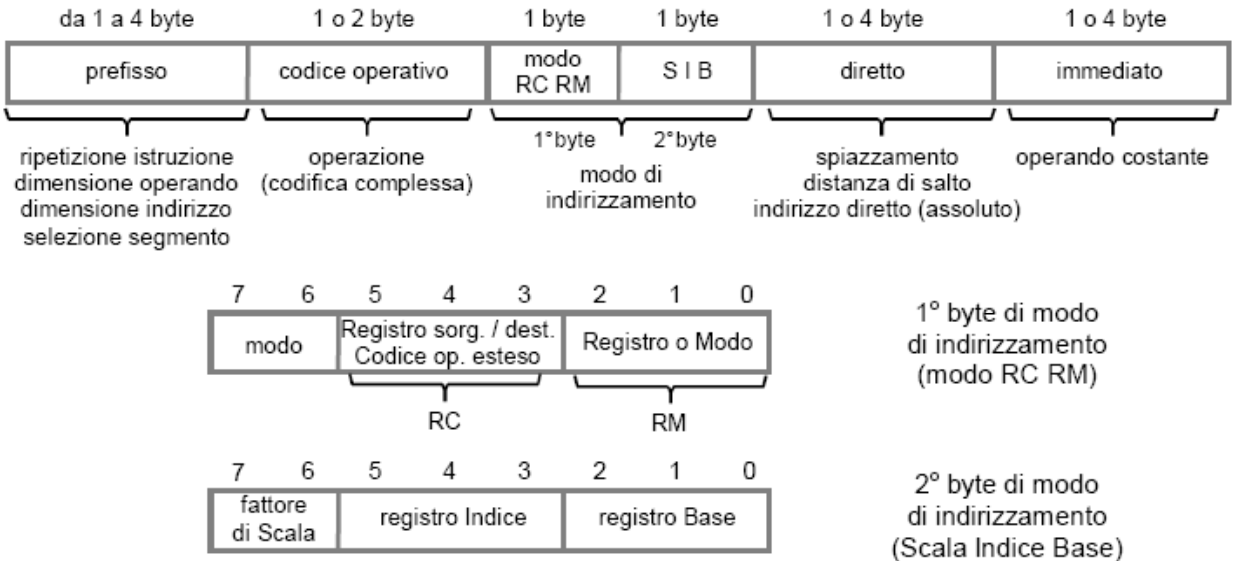



(c) codifica binaria a tre parole

la parola di memoria è da 32 bit (4 byte)

formato adatto se l'istruzione è a due arg. e nomina due indirizzi

# Esempio istruzione architettura IA-32




formato binario di istruzione in linguaggio macchina Intel a 32 bit (Pentium)
molto generale e potente (esprime qualsiasi istruzione)
ma molto complesso e irregolare …



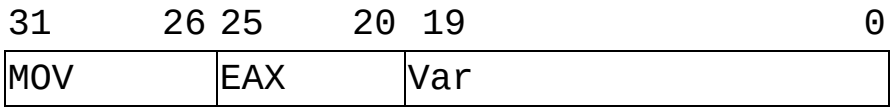
# Esempio di codifica di un'istruzione

**`MOV EAX, Var`**

- EAX è un registro a 32 bit
- `Var` è una posizione di memoria
- il significato di questa istruzione è:
  - copia nel registro EAX il contenuto della cella di 32 bit posta all'indirizzo `Var`; schematicamente:

**`R1 ← Var`**

- Formato dell'istruzione:




- Se il MOV è codificata come 110001
- EAX è codificato come 00001,
- e Var ha l'indirizzo 1023, l'istruzione è:
  `1100 0100 0010 0000 0000 0011 1111 1111`

## Considerazioni sulla codifica delle istruzioni

L'istruzione:

```
11000100001000000000001111111111
```

può essere scritta in formato esadecimale:

```
6610003FF
```

ma è più comoda la rappresentazione in formato mnemonico:

```
MOV EAX, Var
```

ossia:

*operazione destinatario, sorgente*



## I programmi

- **Un programma** è una sequenza di istruzioni (*statement*).
- **Un programma** è la traduzione in uno specifico *linguaggio di programmazione* di un *algoritmo*.
- **L'algoritmo** è un procedimento che risolve in una sequenza finita di passi un problema dato.

- Una sequenza di istruzioni macchina costituisce un programma in linguaggio macchina.

## Dalle istruzioni macchina al linguaggio Assembler

- Un calcolatore elettronico è in grado d'interpretare istruzioni molto semplice come le istruzioni macchina.
- Le istruzioni macchina sono in formato binario, comprensibile alla CPU.
- Per semplicità associamo a queste istruzioni delle rappresentazioni simboliche. La loro sequenza costituisce un programma in linguaggio *Assembler*.



## Generalità sul linguaggio Assembler

- E' il linguaggio di più basso livello.
- Le istruzioni in linguaggio assembler sono in corrispondenza biunivoca con le istruzioni in linguaggio macchina.

  → *Pertanto il programmatore deve ragionare come la CPU.*

# Sintassi del linguaggio Assembler

Ha una sintassi del tipo:

```
LABEL    OP   OPN1, OPN2, …    ;Commento
```

dove:

- `LABEL` è una etichetta simbolica opzionale;
- `OP` è l'operazione in formato mnemonico;
- `OPN1, OPN2, …` sono gli eventuali parametri;
- `;Commento` è l'eventuale commento del programmatore.

Ogni istruzione occupa una riga.



# Dai linguaggi ad alto livello a quelli a basso livello

- Il linguaggio ad alto livello utilizza schemi per risolvere gli algoritmi più vicini alla mente umana ed elimina la dipendenza alla particolare CPU/Architettura utilizzata, in quanto astrae dai dettagli di funzionamento dell'elettronica sottostante.
- I linguaggi ad alto livello devono essere tradotti da opportuni programmi detti compilatori che ne producono i programmi *oggetto*.
- I programmi oggetti vengono collegati tra loro e con le *librerie* da un programma detto *Linker* che produce il file *eseguibile* vero e proprio che è in formato macchina.
- Il file eseguibile viene caricato in memoria da un programma *Loader* che lo passa alla CPU per l'esecuzione.

# Forma Simbolica

- Il programmatore denota l'istruzione macchina in forma simbolica, facilmente leggibile, come per esempio:

  NOME  arg1, arg2, …  commento

- Il nome (o codice mnemonico) indica l'operazione: MOVE (carica, memorizza o copia dato), ADD (addiziona dato), SUB (sottrai), ecc
- Gli elementi arg1, arg2, ecc, sono gli argomenti e indicano i dati da elaborare o dove scrivere il risultato, o anche, nelle istruzioni di salto, dove reperire la prossima istruzione da eseguire.
- Il commento è solo ad uso del programmatore.



# Sorgente e Destinazione

- Se un argomento specifica un dato da elaborare o come reperirlo in memoria o nei registri del processore, si dice di tipo sorgente (di dato).
- Se un argomento specifica dove andare a scrivere il risultato dell'elaborazione (in mem. o in un reg.), si dice di tipo destinazione (di dato).
- Se un argomento specifica dove è collocata in memoria (cioè a quale indirizzo si trova) la prossima istruzione da eseguire, si dice di tipo destinazione di salto.
- Nell'istruzione un argomento può fungere talvolta sia da sorgente sia da destinazione

## Istruzione a Due Argomenti

- Istruzione a due argomenti o binaria:

  NOME  arg1, arg2

- Esempio:

  ADD R1, R2
  - addiziona i contenuti di R1 e R2 e scrivi la somma in R1, sovrascrivendone il contenuto precedente
  - R2 è sorgente, R1 è sia sorgente sia destinazione



## Istruzione a Tre Argomenti

- Istruzione a tre argomenti o ternaria:

  NOME  arg1, arg2, arg3

- Esempio:

  ADD R1, R2, R3
  - addiziona i contenuti di R2 e R3 e scrivi la somma in R1, sovrascrivendone il contenuto precedente
  - R1 e R2 sono sorgente, R3 è destinazione
  - arg3 potrebbe essere identico a arg1 o arg2 !

# Istruzione a un Argomento

- Istruzione a un argomento o unaria:

  NOME arg

- Esempio:

  DEC R1
  - decrementa il contenuto di R1
  - R1 è sia sorgente sia destinazione



# Istruzione a un Argomento (Salto)

- Altro esempio:

  JMP 10
  - sposta il flusso di esecuzione del programma all'istruzione collocata all'indirizzo 10 (e poi và avanti di seguito agli indirizzi 11, 12, ecc)
  - 10 è destinazione di salto
  - JMP (R1)
  - sposta il flusso di esecuzione del programma all'istruzione il cui indirizzo è contenuto in R1 (e poi, partendo da là, và avanti di seguito)
  - R1 è destinazione di salto

## Istruzione senza Argomento

- Istruzione senza argomento o nullaria:

  NOME

- Esempio:

  NOP
  - nessuna operazione, perdi solo un po' di tempo (ce ne sono anche altre, più utili)



## Indirizzo ed Etichetta

- Le istruzioni macchina costituenti il programma sono contenute in memoria, in parole consecutive, e ognuna di esse è identificata da un indirizzo numerico.
- Per comodità, nella forma simbolica scritta dal programmatore, si marcano le istruzioni di interesse tramite un'etichetta, un identificatore simbolico significativo.

# Indirizzo ed Etichetta

- Ecco un'istruzione (binaria) marcata con etichetta, che ne richiama il significato:

  SOMMA: ADD R1, R2

- Per l'assemblatore, il simbolo SOMMA è una costante (simbolica) corrispondente all'indirizzo di memoria della parola che conterrà la codifica numerica di ADD, quando il programma sarà caricato in memoria per l'esecuzione (il separatore ":" è per lo più facoltativo).
- Naturalmente, è vietato duplicare le etichette.



# Dato ed Etichetta

- Le parole di memoria possono anche contenere dati, oltre a istruzioni macchina.
- Si può marcare con un'etichetta simbolica anche una parola di dato, è molto comodo e chiaro.
- Ciò si fa mediante una direttiva, una specie di dichiarazione data all'assemblatore come aiuto per analizzare il programma e tradurlo in forma numerica. La direttiva in questione:
  - specifica che certe parole di memoria sono per dati
  - le marca (una per una o in blocco) con etichette
- L'elenco delle direttive è piuttosto standard e quelle importanti saranno spiegate più avanti.

## Ordinamento Sorgente-Destinazione e Destinazione-Sorgente

- Nel linguaggio macchina della maggior parte delle famiglie di processori, gli argomenti sorgente precedono quello destinazione.
- Prototipi tipici:
  NOME arg_sorg1, arg_sorg_2, arg_dest
  NOME arg_sorg, arg_dest
  NOME arg_sorg, arg_sorg_e_dest
  **NOME arg_dest, arg_sorg**



## Numero di Argomenti

- I linguaggi macchina si dividono in:
  - linguaggi a due argomenti: non ci sono istruzioni a tre argomenti e le istruzioni che fanno operazioni con due operandi e un risultato lo sovrascrivono a uno degli operandi, il quale va dunque perso
  - linguaggi a tre argomenti: le istruzioni che fanno operazioni con due operandi e un risultato non modificano gli operandi e scrivono altrove il risultato
- Entrambe le categorie di linguaggi contengono sempre anche numerose istruzioni unarie e istruzioni senza argomenti, naturalmente.
- Ad esempio: i due linguaggi Motorola e Intel sono a due argomenti.

I modelli di esecuzione delle istruzioni

# TIPI DI ARCHITETTURA



## Tipi di architettura di un calcolare (1/2)

- Consideriamo l'operazione: `a = c + b;`
  i valori a, b, e c sono memorizzati in memoria agli indirizzi A, B, e C.
- Poiché l'elaborazione è effettuata nella CPU è necessario:
  1) leggere la parola all'indirizzo B,
  2) leggere la parola all'indirizzo C,
  3) eseguire la somma tramite l'ALU,
  4) scrivere il risultato all'indirizzo A

→Questa operazione può essere eseguita in modi diversi a seconda dell'architettura utilizzata sul calcolatore.

# Tipi di architettura di un calcolare (1/2)

- I tipi di architettura possono essere classificati in base al modello di esecuzione:
  - Modello a stack
  - Modello Memoria – Memoria
  - Modello Registro – Registro
  - Modello Registro – Memoria



# Modello a Stack

```
PUSH B ; Salva il contenuto di B nello stack
PUSH C ; Salva il contenuto di C nello stack
ADD    ; Somma gli ultimi 2 valori dello stack
POP  A ; Copia il contenuto dell'ultima
         posizione dello stack in A
```

\+ Gli operandi di ADD sono impliciti

\+ Modello utile per alcuni linguaggi ad alto livello;

\- Questo modello comporta un traffico elevato verso la memoria (PUSH e POP comportano due operazioni in memoria, una nella memoria dati e una nella parte stack, e anche ADD deve accedere tre volte alla memoria); considerata la maggior velocità dei registri della CPU questo svantaggio ha fatto abbandonare il modello.

# Modello Memoria – Memoria

```
ADD  A, B, C
```

+ Unica istruzione macchina;
- l'istruzione deve avere un formato molto ampio per contenere tre campi per contenere i tre indirizzi;
- ci sono problemi dovuti al fatto che la memoria effettua una sola operazione alla volta e quindi i tre indirizzi devono essere asseriti in tempi diverse;
  - → *logica di esecuzione complessa*.
- Anche se l'istruzione prevede soli indirizzi di memoria la CPU deve contenere dei registri di appoggio in cui la logica di controllo copia temporaneamente i valori delle variabili lette prima di sommarli.



# Modello Registro – Registro

```
MOV  BX, B      ; Copia il contenuto di B in BX
MOV  CX, C      ; Copia il contenuto di C in CX
ADD  AX, BX,CX  ; Somma BX a CX
                ; e lascia il risultato in AX
MOV  A, AX      ; Copia di contenuto di AX in A
```

+ *Semplice*: ogni passo dell'algoritmo prevede un'istruzione macchina;
+ ogni istruzione indirizza al più una cella di memoria;
- occorrono quattro istruzioni;
- l'istruzione ADD prevede due sorgenti ed un destinatario.

# Modello Registro – Memoria

```
MOV  AX, B      ; Copia il contenuto di B in AX
ADD  AX, C      ; Somma il contenuto di C ad AX
                ; e lascia il risultato in AX
MOV  A, AX      ; Copia di contenuto di AX in A
```

+ Si utilizza un solo registro;
+ ogni istruzione indirizza al più una cella di memoria;
+ l'istruzione ADD prevede un sorgente ed un destinatario, come le altre istruzioni.



# Esempio di allocazione della memoria secondo il modello registro-memoria

| | | | |
|---|---|---|---|
| MOV | AX | B | I |
| ADD | AX | C | I+4 |
| MOV2 | AX | A | I+8 |
| HLT | | | |
| … | | | |
| 9999 | | | B |
| … | | | |
| 777 | | | C |
| … | | | |
| … | | | |
| ? | | | A |
| … | | | |

- Ogni istruzione è eseguita solo dopo che si è conclusa la precedente.
- Esiste un contatore di programma – PC – che avanza al termine dell'esecuzione di ogni istruzione: PC = PC+4.
- Nella fase di *fetch* l'istruzione letta dalla memoria viene portata nel registro IR della CPU.
  - → Quindi avviene l'interpretazione e l'esecuzione di segnali di comando appropriati con le dovute temporizzazioni.
- → La fase di *fetch* è uguale per tutte le istruzioni mentre quella di esecuzione varia da una all'altra.

# Formalismo RTL

*Register Transfer Language (RTL)*
una notazione semplice per specificare
in modo rigoroso come funziona
e che cosa fa l'istruzione macchina

## Problema

- È scomodo, lungo e spesso impreciso spiegare informalmente (a parole, in italiano o inglese, ecc) come funziona un'istruzione macchina.
- Conviene avere una notazione di tipo matematico, formale e precisa, per specificare che cosa faccia un'istruzione.
- RTL: register transfer language, è un formalismo notazionale di specifica per istruzioni.
- Somiglia al costrutto di assegnamento a variabile, come si trova in C, Pascal e simili.

## Simbologia di Base

- Il numero decimale indica:
  - una costante numerica, da usare come dato
  - un indirizzo di memoria, da usare come riferimento a una parola di memoria
- Un identificatore simbolico, magari contenente un suffisso numerico, indica:
  - un nome di registro, come PC, SP o Ri (con suffisso i), il cui contenuto si usa come dato o come indirizzo
  - una costante numerica cui sia stato dato un nome simbolico, come per esempio si fa in C con #define, da usare come dato



## Simbologia di Base

- L'operatore freccia verso sinistra "←" funziona in sostanza come operatore di assegnamento:

  destinazione valore ← origine valore

- Tipicamente l'origine del valore è una costante, un numero contenuto in un registro, una parola di memoria o un'espressione aritmetica tra oggetti di tale genere.
- Somiglia a scrivere in C:

  *variabile = espressione con var. e cost.*

# Operatore di Riferimento

- L'indirizzo è un numero (positivo o al minimo nullo) e anche il dato lo è, o lo si può facilmente ridurre a numero (carattere – codice ASCII).
- Come distinguere tra un numero inteso come dato o come riferimento a un contenitore di dato (parola di memoria o registro) ? Cioè, come capire se il numero è dato o indirizzo ?
- Per indicare che un numero sia da intendere come indirizzo (che si riferisce indirettamente a un dato) e non direttamente come dato da elaborare, racchiudilo tra parentesi quadre [ e ].



# Esempi

- R1 ← 10
  - sovrascrivi in R1 la costante 10 (decimale)
  - il contenuto precedente di R1 va perso
- R1 ← [ R2 ]
  - sovrascrivi in R1 il valore contenuto in R2
  - il contenuto precedente di R1 va perso
  - perché si scrive [ R2 ] e non semplicemente R2 ?
  - perché altrimenti R2 verrebbe preso come costante simbolica, mentre è un nome di registro (un oggetto che ha un contenuto mutevole in corso d'esecuzione)

# Esempi (1/2)

- 10 ← R1
  - sovrascrivi nella parola di memoria di indirizzo 10 (decimale) il contenuto di R1
  - il valore precedente della parola va perso
  - perché non si scrive [ 10 ] ?
  - a sinistra di ← non servono quadre per indicare che 10 è indirizzo e non costante, che senso avrebbe infatti “sovrascrivere una costante” ? (essa non sarebbe più tale se fosse modificabile)
- 10 ← 20
  - sovrascrivi nella parola di memoria di indirizzo 10 la costante 20 (decimale)
  - il contenuto precedente della parola va perso



# Esempi (2/2)

- 10 ← [ 20 ]
  - sovrascrivi nella parola di memoria di indirizzo 10 il valore (il contenuto) della parola di memoria collocata all’indirizzo 20
  - il contenuto precedente della parola di indirizzo 10 va perso
- A sinistra di ← si usa un livello di parentesi in meno rispetto a quanto si debba fare a destra.

# Operazioni

- R1 ← 10 + [ R2 ]
  - addiziona la costante 10 al contenuto di R2 e sovrascrivi la somma in R1
  - il contenuto precedente di R1 va perso
- R1 ← [ 10 ] + [ R2 ]
  - come prima, ma non la costante 10, bensì il contenuto della parola di memoria di indirizzo 10
- Si possono scrivere espressioni più complicate, con addizione, sottrazione, op. logiche, ecc, e operandi di tutti i tipi (si vedano esempi avanti).



# Complicazione

- R1 ← [ [ R2 ] ]
  - sovrascrivi in R1 il contenuto della parola di memoria il cui indirizzo è contenuto in R2
  - R2 funziona come puntatore a memoria !
- [ R1 ] ← [ R2 ]
  - sovrascrivi nella parola di memoria il cui indirizzo è contenuto in R1, il contenuto di R2
  - R1 funziona come puntatore a memoria !
- Note bene: R2 e R1 sono usati come puntatori in lettura e scrittura, rispettivamente, ma il numero di parentesi è diverso (come detto prima).

## Conclusione

- Si presti molta attenzione al significato e all'uso delle parentesi quadre, sono assolutamente essenziali e non vanno dimenticate !
- Le istruzioni macchina spesso vengono commentate dandone l'interpretazione in RTL. Per esempio:

  ADD R2, R1 $\quad\quad$ R2 $\leftarrow$ [ R1 ] + [ R2 ]



Alcune istruzioni semplici
e programmi che ne fanno uso

# ESEMPI SEMPLICI DI ISTRUZIONE

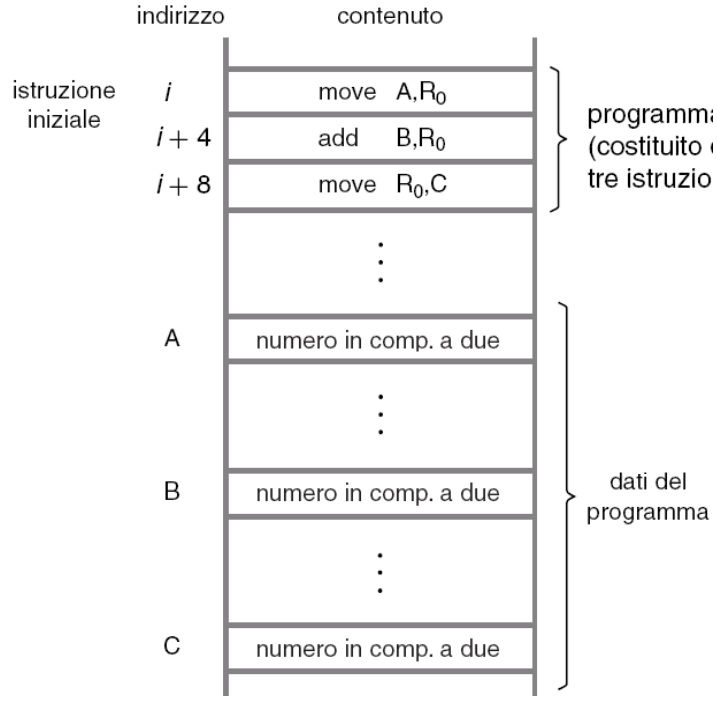
# MOVE e ADD

```
MOVE Ri, A          ; Ri ← [ A ]
ADD  Ri, B          ; Ri ← [ B ]
MOVE C, Ri          ; C ← [ Ri ]
```

- Di seguito si mostra come tradurre in linguaggio macchina il programma in C:
  - int A, B, C;
  - C = A + B;
- Si suppone che il tipo int occupi una parola di memoria (da 16 o 32 bit).



# Addizione di Due Interi (*notazione sorg,dest*)




Programma che esegue l'operazione C ← [A] + [B]

# Come Addizionare una Lista

- Si estende facilmente la soluzione precedente per calcolare la somma di una lista (o un vettore) di n ≥ 2 numeri interi.
- In modo ingenuo, basta replicare le istruzioni macchina di prima cambiandone gli argomenti di tipo indirizzo, e così costruire un programma di maggiore lunghezza benché alquanto ripetitivo come significato.
- Funziona senz'altro ed è semplice, ma naturalmente è anche molto ingombrante.



# Addizione di Lista o Vettore (notaz. sorg,dest)

| | | |
|---|---|---|
| $i$ | move | NUM1,$R_0$ |
| $i+4$ | add | NUM2,$R_0$ |
| $i+8$ | add | NUM3,$R_0$ |
| | ⋮ | |
| $i+4n-4$ | add | NUM$n$,$R_0$ |
| $i+4n$ | move | $R_0$,SOMMA |
| | | |
| | ⋮ | |
| SOMMA | numero in comp. a due | |
| NUM1 | numero in comp. a due | |
| NUM2 | numero in comp. a due | |
| | ⋮ | |
| NUM$n$ | numero in comp. a due | |

Programma sequenziale che calcola la somma di una lista di numeri.

# Come Addizionare una Lista

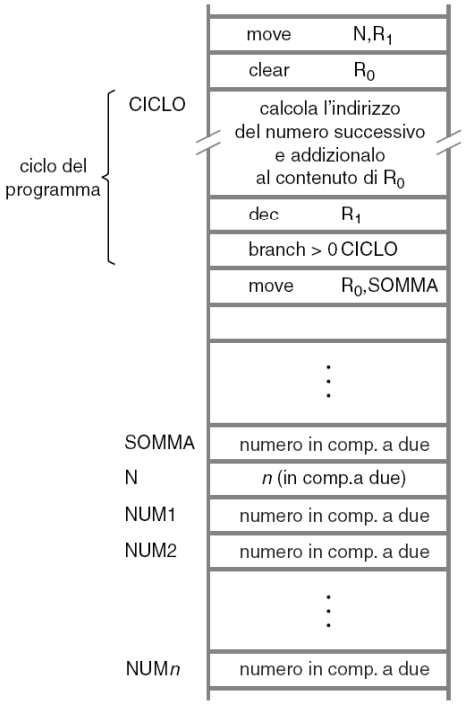
```
CLEAR    Ri        ; Ri ← 0
DEC      Ri        ; Ri ← [ Ri ] – 1
BRANCH > 0 CICLO
```

- se l'operazione precedente dà risultato 0, passa a eseguire l'istruzione all'etichetta CICLO (non quella consecutiva, come di solito); altrimenti vai di seguito
- ossia: PC ← if (> 0) ind. CICLO else [PC] + 2 (ipotesi: l'istruzione BRANCH occupa due byte)

- Mediante queste nuove istruzioni, il programma di prima diventa un ciclo, molto meno ingombrante e concettualmente più significativo.



# Addizione di Lista con Ciclo




Programma ciclico che calcola la somma di una lista di numeri.

Gruppi principali di istruzioni macchina

# CLASSI DI ISTRUZIONE



## Classi di Istruzione

- Le istruzioni macchina di un generico linguaggio si ripartiscono in poche classi.
- Ogni classe ha una funzione generale ben definita e le singole istruzioni componenti la classe si somigliano molto.
- Spesso basta spiegare pochi esempi di istruzione per classe, e tutte le altre si comprendono semplicemente leggendole.

# Elenco delle Classi (1/3)

- Istruzioni di trasferimento:
  - caricamento, memorizzazione e copia
- Istruzioni aritmetiche e logiche:
  - di base: addizione, sottrazione, cambio segno, AND, OR, NOT, XOR, e poco altro
  - ausiliarie: scorrimento, rotazione, varie
- Istruzioni di ingresso e uscita:
  - lettura o scrittura di dato da o su periferica



# Elenco delle Classi (2/3)

- Istruzioni di confronto:
  - confronto tra due dati (uguale, diverso, minore, maggiore, minore o uguale, maggiore o uguale) o di un dato con zero
  - esame di bit o di gruppo di bit
- Istruzioni di manipolazione dati complessi:
  - gestione pila (immetti in pila e estrai dalla pila)
  - talvolta altro (coda, ecc)

## Elenco delle Classi (3/3)

- Istruzioni di controllo del flusso di esecuzione del programma (salto):
  - salto incondizionato
  - salto condizionato
  - salto a conteggio (per ciclo)
  - salto a sottoprogramma (routine)
- Istruzioni di controllo del processore:
  - gestione del meccanismo di interruzione
  - chiamata e supervisore (supervisor call, SVC)
  - ecc (altre istruzioni più o meno specialistiche)
- Questa è la classe più disomogenea e di solito contiene poche istruzioni (però fondamentali).



## Breve Elenco di Nomi di Istruzione (1/4)

- MOVE
  - copia dato da sorgente a destinazione
- LOAD
  - carica dato da memoria in registro
- STORE
  - memorizza dato da registro (in memoria)
- Nota bene: MOVE da sola assolve anche le funzioni di LOAD e STORE.
- I nomi LOAD e STORE sono presenti nel linguaggio solo in casi particolari (vedi testo).

## Breve Elenco di Nomi di Istruzione (2/4)

- ADD
  - addizione di due numeri (interi o in comp. 2)
- SUB
  - sottrazione di due numeri (idem)
- CMP
  - confronto di due numeri (idem): uguale, diverso, minore, maggiore, minore o uguale, maggiore o uguale



## Breve Elenco di Nomi di Istruzione (3/4)

- OR
  - somma logica bit a bit (bitwise) di due parole
- AND
  - prodotto logico bit a bit (bitwise) di due parole
- NOT
  - complemento logico (inversione, negazione) bit a bit (bitwise) di una parola
- Talvolta altre, come XOR (o EOR), ecc.

## Breve Elenco di Nomi di Istruzione (4/4)

- BRANCH CC (BCC)
  - salto condizionato, dove “CC” indica una condizione (≠, =, <, >, ≤, ≥)
- JUMP (JMP)
  - salto, di solito incondizionato
- CALL e RETURN
  - chiamata a sottoprogramma e rientro da sottoprogramma



# Modi di indirizzamento

Come l’istruzione macchina reperisce i dati da elaborare e determina la prossima istruzione da eseguire

# Modo di Indirizzamento

- Il modo di indirizzamento è un insieme di regole secondo cui denotare nell'istruzione macchina come:
  - reperire il dato da elaborare o la posizione dove andare a scrivere il risultato dell'elaborazione – modo di dato
  - individuare la prossima istruzione da eseguire, quando non si deve andare di seguito (caso dell'istruzione di salto) – modo di istruzione
- In teoria, i modi di dato e di istruzione sono interscambiabili, ma ci sono alcuni usi prevalenti.
- I modi di indirizzamento sono essenziali nel dare flessibilità e generalità al linguaggio macchina.
- Se sono numerosi sono però anche complessi da gestire e complicano la descrizione del linguaggio stesso.



# Elenco dei Modi

| Modo | Sintassi | Funzionamento (in RTL) |
|---|---|---|
| immediato (o di costante) | # valore | operando = valore |
| di registro | $R_i$ | operando = $[R_i]$, risultato = $R_i$ |
| assoluto (o diretto) | indirizzo | i.e. = indirizzo |
| indiretto da registro | $(R_i)$ | i.e. = $[R_i]$ |
| indiretto da memoria | (indirizzo) | i.e. = [indirizzo] |
| con indice e spiazzamento | spi. $(R_i)$ | i.e. = spi. + $[R_i]$ |
| con base e indice | $(R_i, R_j)$ | i.e. = $[R_i] + [R_j]$ |
| con base indice e spiazzamento | spi. $(R_i, R_j)$ | i.e. = spi. + $[R_i] + [R_j]$ |
| relativo a contatore di programma | spi. (PC) | i.e. = spi. + [PC] |
| con autoincremento | $(R_i)+$ | i.e. = $[R_i]$ e poi incrementa $R_i$ |
| con autodecremento | $-(R_i)$ | decrementa $R_i$ e poi i.e. = $[R_i]$ |

## Notazione

**Simboli a sinistra di "=":**

"operando" è il *dato* che l'istruzione userà (argomento di tipo sorgente)

"risultato" è il *dato* che l'istruzione produrrà (argomento di tipo destinazione)

"i.e." indica l'*indirizzo effettivo* di memoria dell'argomento (sorgente o destinazione) su cui lavorerà l'istruzione; nell'istruzione di salto indica l'*indirizzo di destinazione*

**Simboli a destra di "=":**

"valore" è un *numero intero relativo*, dotato di segno, con funzione di *costante*

"indirizzo" è un *indirizzo assoluto*, positivo o nullo, riferito alla *memoria*

"$R_i$, $R_j$" sono *registri del processore*, con funzione di *base* e *indice*, rispettivamente

"spi." è un *numero intero relativo*, dotato di segno, con funzione di *spiazzamento*



## Modo Immediato (o di Costante)

```
MOVE  Ri, cost,    ; Ri ← cost
MOVE  R1, #17      ; R1 ← 17 (costante)
```

- La sorgente è una costante, incorporata nell'istruzione e non modificabile (se non riscrivendo l'istruzione …).
- Modo tipico per inizializzare un registro o una parola di memoria, o quando c'è un'operazione aritmetica con una costante. Per esempio:

  ```
  ADD R1, #10        ; R1 ← 10 + [R1]
  ```

- Modo esclusivamente di dato !
- Il prefisso "#" caratterizza il modo immediato.

# Modo di Registro

```
MOVE Rj, Ri        ; Rj ← [Ri]
MOVE R2, R1   ; R2 ← [R1]
```

- il contenuto di R1 viene copiato in R2

- La sorgente è il contenuto di un registro, il cui nome è specificato nell'istruzione.
- Modo tipico per copiare da un registro in un altro, per movimentare o duplicare dati, usatissimo anche per aritmetica. Per esempio:

  ```
  ADD R2, R1         ; R2 ← [R1] + [R2]
  ```

  addiziona i contenuti di R1, R2 e scrivi la somma in R2



# Modo Assoluto (o Diretto)

- `MOVE Ri,IND        ; Ri ← [IND]`
  - il contenuto della cella, o parola, di memoria di indirizzo IND viene caricato in Ri
- `MOVE IND, Ri       ; IND ← [Ri]`
  - il contenuto del registro Ri viene memorizzato nella cella, o parola, di indirizzo IND
- Modo tipico per caricare o memorizzare dati tra processore e memoria centrale.
- Talvolta usato anche per aritmetica. Per esempio:

  ```
  ADD IND, R1          ; R1 ← [IND] + [R1]
  ```

  addiziona i contenuti di R1 e della parola di memoria di indirizzo IND, e scrivi la somma in R1
- Modo prevalentemente di dato, ma usato anche per istruzione (salto), se IND si riferisce a istruzione.

# Modo Indiretto da Registro

- `MOVE Rj,(Ri)  ; Rj ← [[Ri]]`
  - il contenuto della cella, o parola, di memoria puntata da Ri (cioè il cui indirizzo è in Ri) viene caricato in Rj
  - il registro Ri funziona come puntatore !
- `MOVE (Rj),Ri  ;[Rj] ← [Ri]`
  - il contenuto del registro Ri viene memorizzato nella cella, o parola, puntata da Rj
  - attenti, in RTL, all'uso delle parentesi `[` e `]`.
- Modo prevalentemente di dato, ma usato anche per saltare in modo indiretto, se il registro puntatore contiene un indirizzo di istruzione.



# Modo Indiretto Da Memoria

- `MOVE Rj,(IND)  ; Rj ← [[IND]]`
  - il contenuto della cella, o parola, di memoria il cui indirizzo si trova nella cella (parola) di memoria di indirizzo IND, viene caricato in Rj
  - la cella, o parola, di memoria all'indirizzo IND funziona come puntatore !
- `MOVE (IND),Ri  ; [IND] ← [Ri]`
  - analogo all'indirizzamento indiretto da registro …
  - attenti, in RTL, all'uso delle parentesi `[` e `]`.
- Modo prevalentemente di dato, talvolta di istruzione (come prima).

## Modo con Indice e Spiazzamento

- `MOVE Rj,spi.(Ri); Rj ← [spi + [Ri]]`
  - il contenuto della cella, o parola, di memoria il cui indirizzo si ottiene addizionando il contenuto di Ri alla costante spi (incorporata nell'istruzione), viene caricato in Rj
- Modo sia di dato sia di istruzione, nella forma relativa a contatore di programma dove al posto del registro Ri si usa il registro PC (vedi di seguito).



## Modo con Base e Indice

- `MOVE Rk,(Ri, Rj) ; Rk ← [[Ri + Rj]]`
  - il contenuto della cella, o parola, di memoria il cui indirizzo si ottiene addizionando i contenuti di Ri (registro base) e Rj (registro indice) viene caricato in Rk
- Modo prevalentemente di dato, talvolta di istruzione.

## Modo con Base Indice e Spiazzamento

- `MOVE Rk,spi(Ri,Rj) ;Rk ← [spi[Ri+Rj]]`
  - combinazione dei due modi precedenti
- Modo prevalentemente di dato, talvolta di istruzione.



## Modo Relativo a PC

- `JMP etichetta ; PC ← [PC] + distanza da etichetta`
  - la prossima istruzione da eseguire sarà quella marcata simbolicamente tramite etichetta
  - al registro PC viene addizionato un numero (positivo o negativo) esprimente la distanza (in avanti o indietro) tra l'istruzione corrente e quella di destinazione di salto
- Modo prevalentemente di istruzione, talvolta anche di dato (ma poco comune in tale uso).
- Modo tipico delle istruzioni di salto, anche condizionato.
- Vedi anche modo con indice e spiazzamento.

# Modo con Autoincremento

- `MOVE Rj,(Ri)+ ;`

  ```
  prima Rj ←[[Ri]]
  poi   Ri ←[Ri] + dim. dato in byte
  ```

  - come modo indiretto da registro,
    ma dopo l'uso incrementa il registro puntatore Ri
  - in MOVE, realizza l'istruzione POP:
    - POP Rj  estrae il dato dalla pila e lo scrive in Rj
      supponendo che Ri abbia ruolo di SP e che la pila decresca verso gli indirizzi alti.



# … e con Autodecremento

- `MOVE –(Rj),Ri  ;`

  ```
  prima Rj ←[Rj] – dim. dato in byte
  poi [Rj] ←[Ri]
  ```

  - come modo indiretto da registro, ma prima dell'uso decrementa il registro puntatore Rj
  - in MOVE, realizza l'istruzione PUSH:
    - PUSH Ri  mette nella pila il dato contenuto in Ri
      supponendo che Rj abbia ruolo di SP e che la pila cresca verso gli indirizzi bassi

# Ortogonalità di Istruzione

- Se in un'istruzione macchina, per esempio a due argomenti o binaria, questi si possono mettere in un modo di indirizzamento qualunque (purché abbia senso logico secondo l'operazione da eseguire), il primo indipendentemente dal secondo, l'istruzione si dice ortogonale (nel senso di indipendente).
- In alcuni processori l'istruzione MOVE è del tutto o quasi ortogonale.
- Le altre istruzioni in genere presentano restrizioni su come scegliere e combinare i modi di indirizzamento degli argomenti.



Come funziona il salto condizionato

# BIT DI ESITO (O CODICE DI CONDIZIONE)

# Bit di Esito

- Alcune istruzioni macchina, tipicamente aritmetiche (ADD, SUB, ecc), oltre a produrre un risultato, hanno anche un esito (o condizione), che esprime un'informazione (di tipo binario, cioè vero o falso, sì o no) collaterale, o aggiuntiva, al risultato vero e proprio.
- Ecco alcuni esiti molti comuni nei processori:
  - se il risultato sia positivo o negativo
  - se sia nullo o diverso da zero
  - in caso di addizione, se il risultato abbia dato luogo a riporto in uscita (di sottrazione, prestito in uscita)
  - e altri esiti ancora, più specialistici …



# Bit di Esito

- Ciascun esito è rappresentato da un bit:
  - se il bit vale uno, l'esito associato è affermativo
  - altrimenti, l'esito associato è negativo
- Di norma i bit di esito sono collocati nel registro di stato, in posizioni prefissate.
- Le istruzioni macchina che producono esito aggiornano in modo automatico i bit di esito di loro pertinenza.
- Le istruzioni macchina di salto condizionato controllano il bit di esito di loro pertinenza per decidere se la condizione di salto sia vera o falsa, e se saltare o meno come conseguenza.

# Bit di Esito

- bit N: esito negativo (per operazioni in complemento a 2)
  - se N = 1: risultato negativo
  - se N = 0: risultato positivo o nullo
- bit Z: esito zero
  - se Z = 1: risultato nullo
  - se Z = 0: risultato diverso da zero
- bit C: riporto (carry) in uscita (per operazioni in naturale)
  - se C = 1: risultato con riporto in uscita
  - se C = 0: risultato senza riporto in uscita

  Vale solo per operazioni aritmetiche.
- bit V: trabocco (overflow) (per operazioni in C2)
  - concettualmente funziona come l'esito C, ma per operazioni con dati in complemento a due

  Vale solo per operazioni aritmetiche.
- Altri ancora, ma di carattere specialistico …



# Registro di stato

- Registro PSW – *Processor Status Word*
  - i cui bit rappresentano il risultato di operazioni, e
  - il cui contenuto può essere esaminato da istruzioni di salto condizionato.
- Bit/Condizioni contenuti:
  - Z (zero) → il risultato dell'operazione è 0;
  - S (segno) → il risultato dell'operazione è negativo;
  - O (overflow) → l'operazione ha prodotto traboccamento;
  - C (carry) → l'operazione ha prodotto riporto.
- Nell'esempio precedente la `CMP` effettua la differenza tra i due contenuti e valuta il bit Z.

## Salti (*jump*) e diramazioni (*branch*)

- Esistono per permettere agli algoritmi di avere alternative e cicli.
- **Salto incondizionato**

```
JMP Destinazione ; PC ← Destinazione
```

- **Salto condizionato**

```
CMP AX, BX ; Confronto (esempio)
JE  Destinazione ; PC ← Destinazione
```



## Salto Condizionato

```
BRANCH = 0          ETICHETTA
```

- se il bit Z indica esito nullo (cioè se Z = 1)
  `PC ← [PC] + distanza da ETICHETTA`
  altrimenti
  `PC ← [PC] + dim. in byte di BRANCH`
- Vale a dire: l'istruzione BRANCH CC (qui CC è la condizione = 0), esamina il bit di esito Z e su tale base decide se saltare a destinazione (qui indicata simbolicamente tramite etichetta) oppure se andare di seguito (cioè all'istruzione consecutiva a BRANCH).

# Salto Condizionato

- Alcune condizioni (più o meno complesse) richiedono l'esame di due o più bit di esito:
  - la condizione $> 0$ equivale all'esame composito seguente:
    - N = 0 (non negativo) e Z = 0 (non nullo)
- Pertanto, l'istruzione:

  `BRANCH > 0`

  esamina (in modo automatico) due bit di esito, N e Z, e decide di conseguenza.



# Simbologia

- Nei linguaggi macchina specifici dei processori, le varie condizioni sono indicate con acronimi (in inglese), più o meno standard, per esempio:
  - BRANCH = 0 (salta se uguale a 0)
    - ⇒ BEQ (branch if equal to 0)
    - ⇒ JE (Jump if Equal)
  - BRANCH > 0 (salta se maggiore di 0)
    - ⇒ BGT (branch if greater than 0)
    - ⇒ JG (Jump if Greater)
  - BRANCH ≥ 0 (salta se maggiore di o uguale a 0)
    - ⇒ BGE (branch if greater than or equal to 0)
    - ⇒ JGE (Jump if Greater or Equal

## Salto Incondizionato

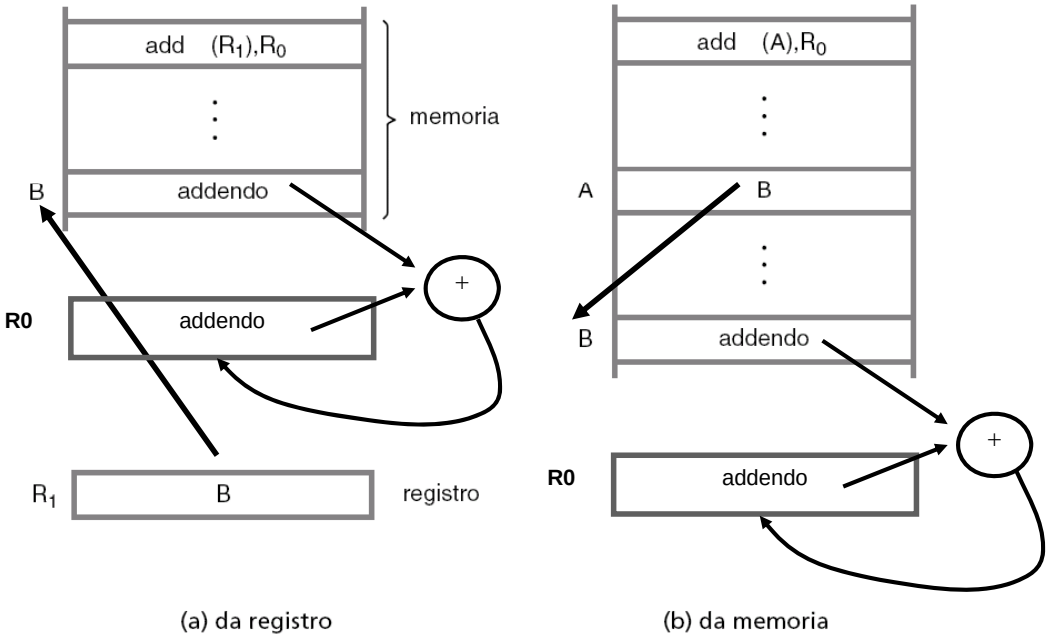
- `JMP ETICHETTA`
  `; PC ← [PC] + ingombro in byte di JMP`
- Il salto avviene sempre, senza alcuna condizione.
- Equivale a supporre la condizione di salto sempre vera.
- Di solito il nome è JMP (o qualcosa di simile) per distinguere dal caso condizionato.
- In certi linguaggi macchina, si scrive JE per BEQ, JGT per BGT, JGE per BGE, ecc …



Esempi vari con modi di indirizzamento

# ALCUNI ESEMPI DI PROGRAMMA

# Indirizzamento Indiretto






# Ciclo a Conteggio e Vettore

```
int N = ...;          -- dim. vettore
int s, i;             -- somma e indice
int NUM [ ... ];      -- vettore di interi
s = 0;                -- inizializza somma
i = N;                -- inizializza dim.
do {                  -- ciclo a conteggio
  s = s + NUM [i];    -- accumulo elemento
  i--;                -- conteggio elemento
} while (i > 0);      -- condizione di ciclo
N.B: si suppone il primo elemento sia NUM[1]
```

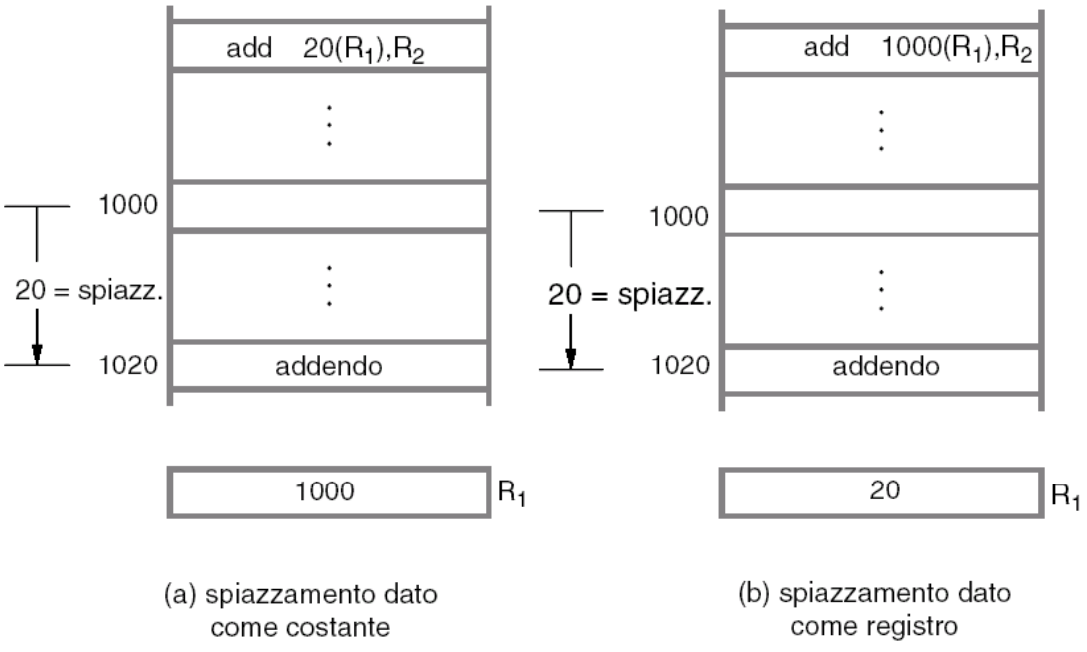
# Ciclo a Conteggio e Vettore (not. sorg.,dest.)

| indirizzo | istruzione | argomento / i | commento |
|---|---|---|---|
| | move | $N,R_1$ | prologo |
| | move | $\#NUM_1,R_2$ | |
| | clear | $R_0$ | |
| CICLO | add | $(R_2),R_0$ | corpo del ciclo |
| | add | $\#4,R_2$ | |
| salto indietro | dec | $R_1$ | |
| | branch > 0 | CICLO | condizioni di ciclo |
| | move | $R_0$,SOMMA | epilogo |

Come usare il modo di indirizzamento indiretto da registro
per calcolare la somma della lista di numeri.



# Indirizzamento Indiretto con Indice e Spiazzamento (notaz. sorg.,dest.)




(a) spiazzamento dato come costante

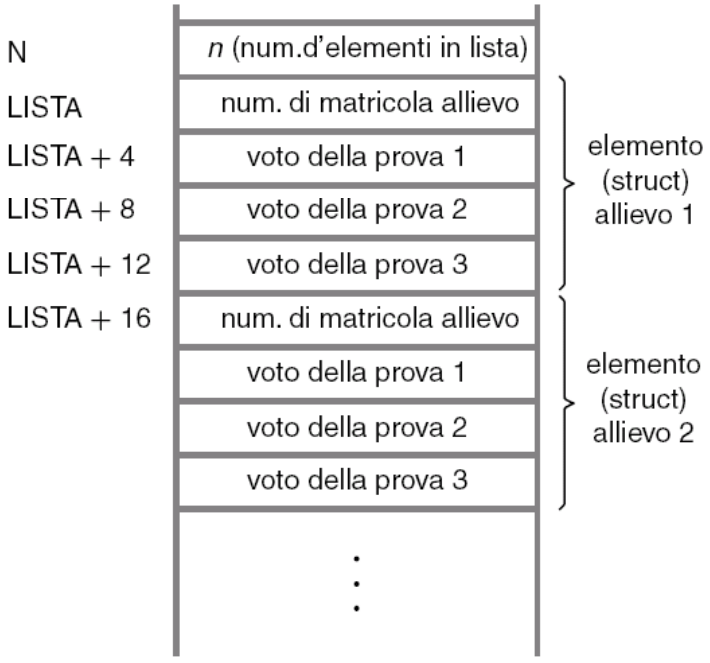
(b) spiazzamento dato come registro

# Vettore di Record (*struct*)

```
typedef struct {  -- tipo dato
  int matricola;      -- campo id
  int voto1;          -- esito prova 1
  int voto2;          -- esito prova 2
  int voto3;          -- esito prova 3
} SCHEDA;             -- nome tipo
int N = ...;          -- dim. vettore
SCHEDA LISTA [...];   -- vettore
```

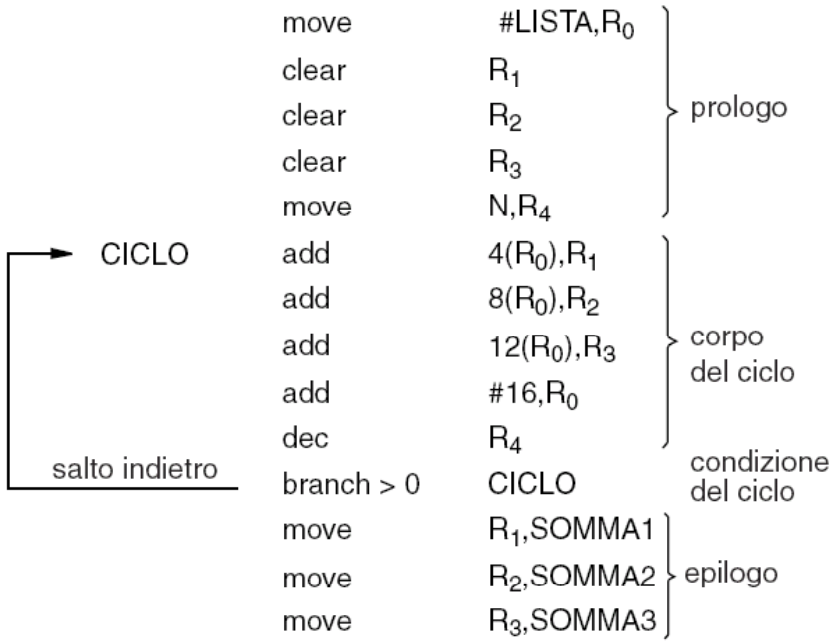


# Vettore di Record (struct)




Elenco studenti con dati relativi alla carriera didattica.

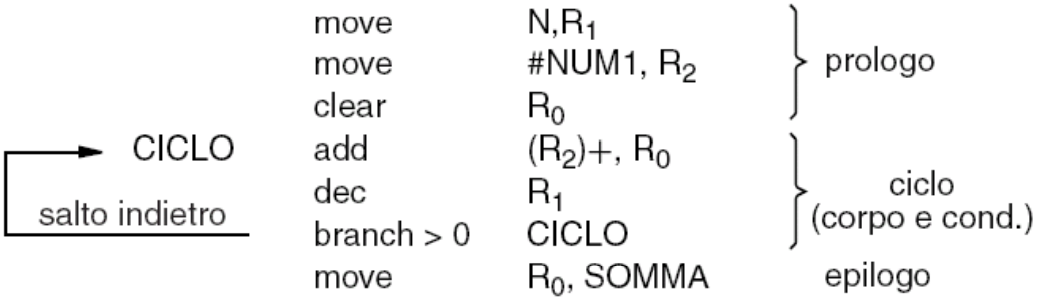
# Programma Addizione Voti (notaz. sorg.,dest.)




Come usare il modo di indirizzamento indiretto da registro
per gestire l'elenco di studenti con dati di carriera didattica.



# Programma Addizione Voti con Autoincremento (notaz. sorg.,dest.)




Come usare il modo con autoincremento
per calcolare la somma della lista di numeri.

istruzioni per comunicare
con i dispositivi di I / O (periferiche)

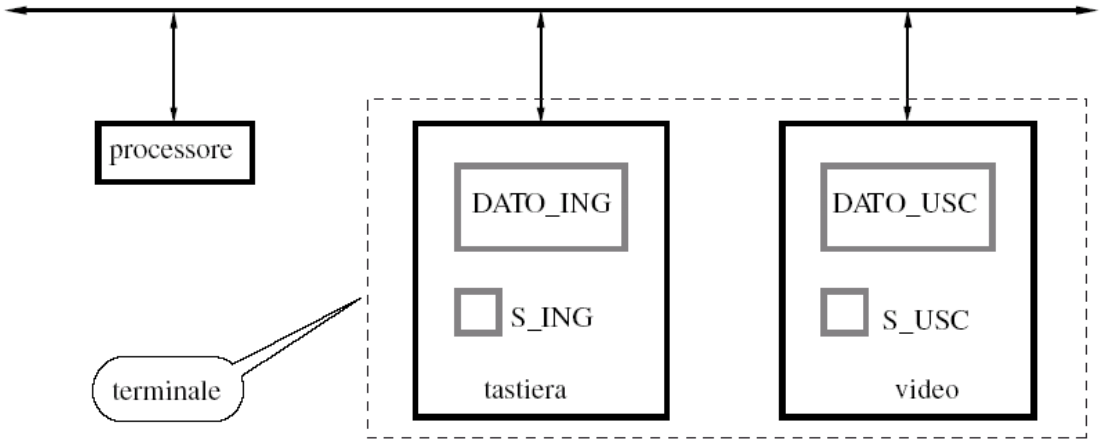
# INGRESSO E USCITA (I / O)



## Interfaccia di I/O

- I dispositivi di I/O (periferiche) vengono visti tramite interfacce di I/O.
- L'interfaccia associa a ciascun dispositivo uno o più indirizzi (come quelli di memoria).
- Per comunicare con il dispositivo, basta leggere o scrivere (secondo il caso) all'indirizzo associato.
- Funziona come caricare (leggere) o memorizzare (scrivere) un dato di memoria.
- Naturalmente, dispositivi diversi devono avere indirizzi differenti (per evitare conflitti).

# Tastiera + Video = Terminale




Calcolatore dotato di bus singolo con unità processore
e interfacce di I / O per tastiera e video.



# Registri di I/O

- DATO_ING: registro (e suo indirizzo simbolico) da dove il processore legge il codice (ASCII) di tasto premuto
- DATO_USC: registro (e suo indirizzo simbolico) dove il processore scrive il codice (ASCII) del carattere da visualizzare (alla posizione corrente del cursore video)
- S_ING: registro (e suo indirizzo simbolico) da dove il processore rileva (legge) un bit di stato:
  - quando vale 1 indica che la tastiera ha pronto un dato non ancora letto
- S_USC: registro (e suo indirizzo simbolico) da dove il processore rileva (legge) un bit di stato:
  - quando vale 1 indica che il video è pronto a riceverne e visualizzare un carattere

# Programma di I/O (not. sorg.,dest.)

PRIMA PARTE - TASTIERA

| etichetta | istruzione | | commento |
|---|---|---|---|
| | move | #BLOCCO, $R_0$ | inizializza il registro $R_0$ facendolo puntare alla testa del blocco di byte in memoria dove scrivere i caratteri introdotti da tastiera |
| TASTIERA | test bit<br>branch = 0 | #3, STATO_ING<br>TASTIERA | attendi la pressione del tasto e l'introduzione del carattere nel registro di I / O DATO_ING monitorandone il bit di stato S_ING (il bit $b_3$ del registro di stato) |
| | move byte | DATO_ING, ($R_0$) | scrivi nella parola di memoria puntata da $R_0$ il carattere contenuto nel registro di I / O DATO_ING (così il bit di stato S_ING della tastiera si azzera automaticamente) |



# Programma di I/O (not. sorg.,dest.)

SECONDA PARTE - VIDEO

| | | | |
|---|---|---|---|
| | | | azzera automaticamente) |
| VIDEO | test bit<br>branch = 0 | #3, STATO_USC<br>VIDEO | attendi che il video sia pronto monitorandone il bit di stato S_USC (il bit $b_3$ del registro di stato) |
| | move byte | ($R_0$), DATO_USC | scrivi nel registro di I / O DATO_USC il carattere puntato da $R_0$ (così il bit di stato S_USC del video si azzera automaticamente) |
| | compare byte<br>branch ≠ 0 | #CR, ($R_0$) +<br>TASTIERA | verifica se il carattere appena letto da tastiera sia CR (ritorno a capo): se non lo è salta indietro e attendi la nuova introduzione; in ogni caso incrementa il registro puntatore $R_0$ |

La pila di memoria (memory stack)
come è fatta e le istruzioni per gestirla

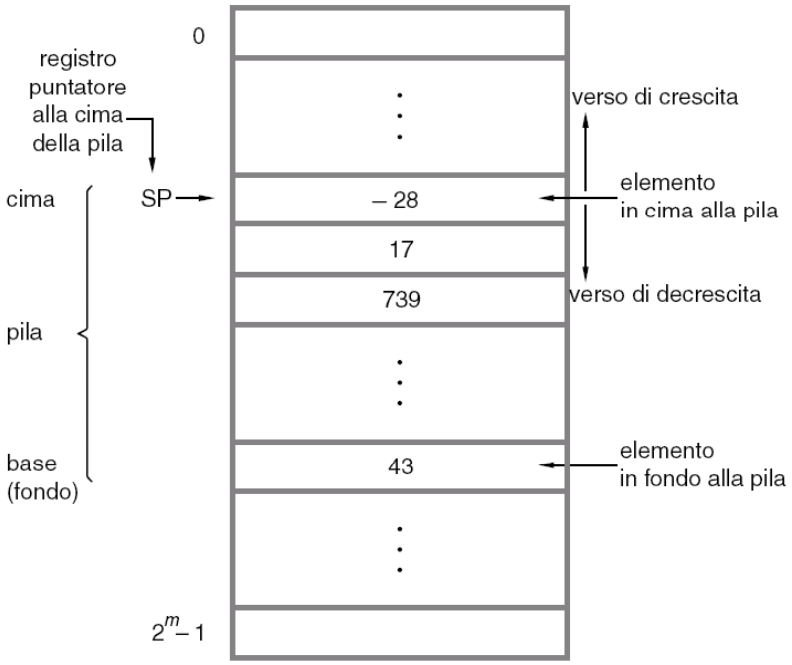
# STRUTTURA E USO DELLO *STACK*



## Pila di Memoria (Stack)

- Struttura dati usata principalmente per la gestione di sottoprogramma.
- Successione di parole (celle) di memoria consecutive.
- Si inserisce una parola solo sulla cima.
  - operazione PUSH (di una parola in pila)
- Si toglie una parola solo dalla cima.
  - operazione POP (di una parola dalla pila)

## Struttura e Terminologia




Struttura della pila di memoria.

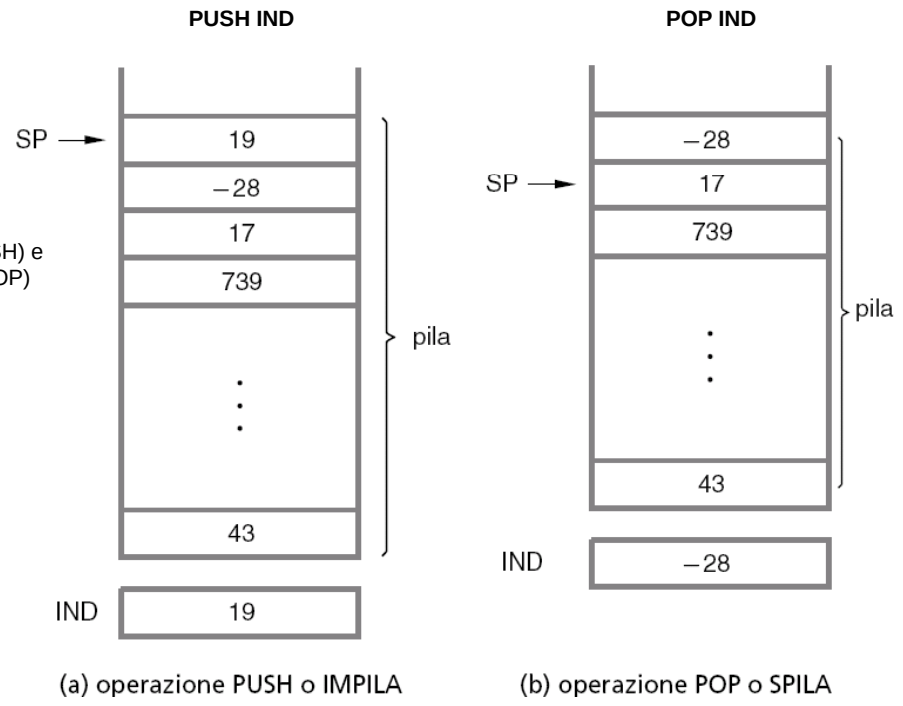


## Operazioni di *stack*

- Una pila è una struttura dati in cui i dati vengono inseriti e prelevati da una estremità – *l'ultimo arrivato è il primo servito*.
- Lo stack è diffuso in molte architetture ed acceduto da due operazioni:
  - PUSH → immette i dati nello stack;
  - POP → preleva i dati dallo stack.
- Di solito si utilizzano due registri per la sua gestione:
  - BP → utilizzato per indirizzare all'interno dello stack;
  - SP → utilizzato per indirizzare la testa dello stack.

# Operazioni di Pila

si può anche immettere in pila (PUSH) e estrarre dalla pila (POP) da un registro o in un registro






# PUSH con Autodecremento (not. sorg.,dest.)

| etichetta | istruzione | | commento |
|---|---|---|---|
| IMPILA | compare | #1500, SP | confronta con 1500 il valore del puntatore SP alla cima della pila |
| | branch ≤ 0 | ERR_PIENA | se è minore o uguale a 1500 la pila è piena; in tale caso passa a eseguire la routine di errore che inizia all'indirizzo ERR_PIENA |
| | move | IND, −(SP) | altrimenti aggiorna il puntatore SP, decrementandolo e poi copia sulla cima della pila la parola di memoria di indirizzo IND (vale a dire impila l'elemento memorizzato nella parola di indirizo IND) |

**equivale a**

**PUSH IND**

(a) operazione IMPILA o PUSH

## POP con Autoincremento (not. sorg.,dest.)

| etichetta | istruzione | | commento |
|---|---|---|---|
| SPILA | compare | #2000, SP | confronta con 2000 il valore del puntatore SP alla cima della pila |
| | branch > 0 | ERR_VUOTA | se è maggiore di 2000 la pila è vuota; in tale caso passa a eseguire la routine di errore che inizia all'indirizzo ERR_VUOTA |
| | move<br>equivale a<br>POP IND | (SP) +, IND | altrimenti copia l'elemento in cima alla pila nella parola di memoria di indirizzo IND e poi aggiorna il puntatore SP incrementandolo (vale a dire spila memorizzando l'elemento nella parola di indirizzo IND) |

(b) operazione SPILA o POP



## Sottoprogrammi

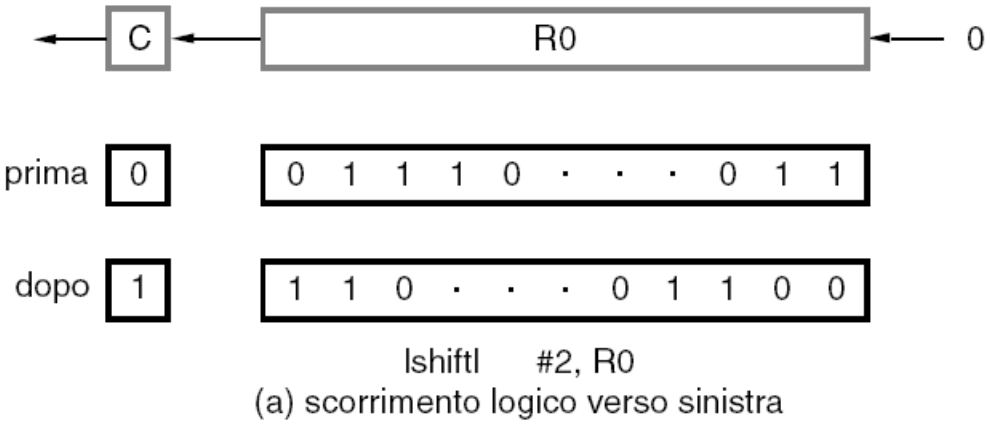
- I sottoprogrammi sono blocchi di istruzioni in linguaggio macchina.
- La chiamata al sottoprogramma rappresenta il trasferimento del controllo al sottoprogramma.
- Al termine del sottoprogramma il controllo ritorna all'istruzione successiva a quella della chiamata.
- Lo stack è utilizzato:
  - effettuare il collegamento tra programma chiamante e sottoprogramma;
  - conservare l'indirizzo di ritorno;
  - attuare il passaggio dei parametri.

istruzioni macchina varieper aritmetica logica e altro

# ALTRE ISTRUZIONI MACCHINA



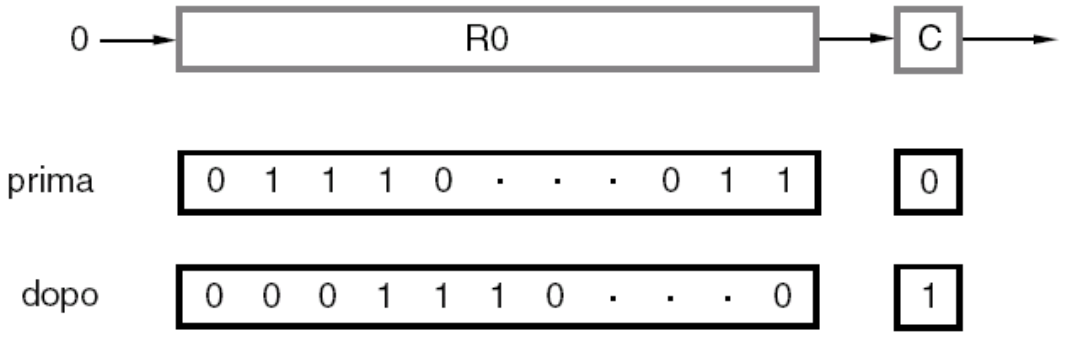
## Scorrimento Logico verso SX




lshiftl #2, R0

(a) scorrimento logico verso sinistra

# Scorrimento Logico verso DX

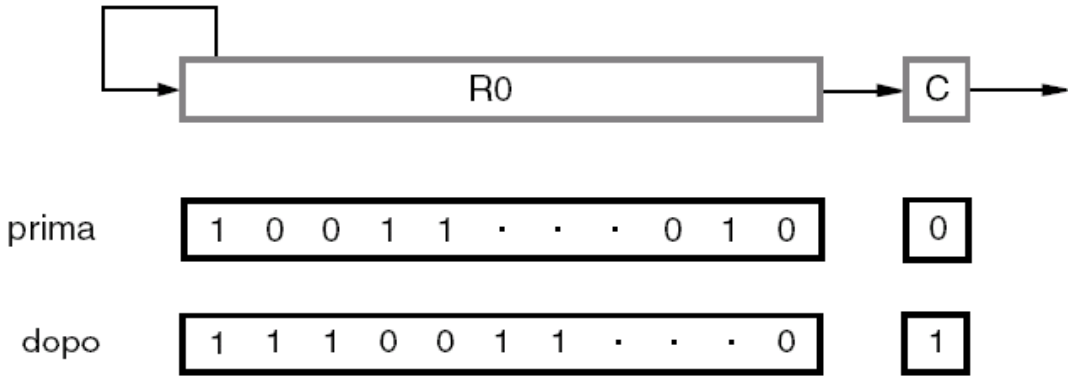



lshiftr #2, R0

(b) scorrimento logico verso destra



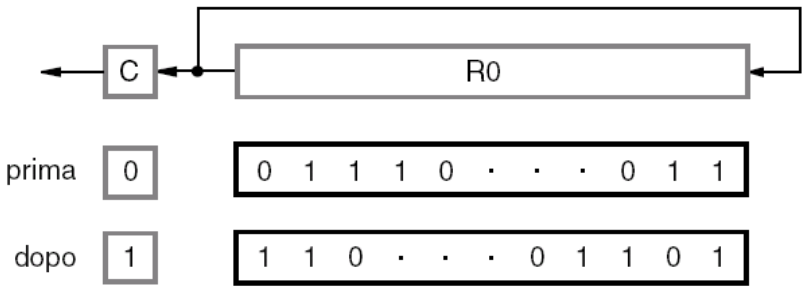
# Scorrimento Aritmetico verso DX




ashiftr #2, R0

(c) scorrimento aritmetico verso destra

# Rotazione verso SX




rotatel #2, R0
(a) rotazione verso sinistr

e similmente tutte le altre rotazioni ...
(vedi testo)



# Le interruzioni

- Esse sono eventi che alterano il *normale* flusso di esecuzione di un programma imponendo alla CPU di abbandonarlo per passare ad eseguire un altro programma dipendente dalla natura dell'interruzione.

## Classificazione dell'interruzioni

- **Esterne**: si verificano in modo *asincrono* rispetto all'esecuzione del programma, comunemente gestiscono operazioni di I/O.
- **Eccezioni**: si verificano in modo *sincrono* rispetto all'esecuzione del programma e *non sono predicibili* in quanto dipendono dallo stato della macchina (ad es. divisioni per zero, *page fault*, ecc.).
- **Trappole** (*Traps*): sono generate da apposite istruzioni presenti nel programma e quindi sono *sincrone* e *predicibili*: producono effetti simili alle istruzioni di salto.



## Istruzioni Varie

- Istruzioni macchina per lavorare su bit singoli:
  - verificare (*test*)
  - portare a 0 (*reset*)
  - portare a 1 (*set*)

  un bit in posizione specifica in un registro o in una parola di memoria.
- Aritmetica avanzata: moltiplicazione, divisione, resto, e altro ancora.
- Si veda il libro o i linguaggi macchina specifici dei processori commerciali.

# Repertorio delle istruzioni

Le prestazioni dei processori sono influenzate da numero istruzioni, potenzialità, formato.

Formati del repertorio delle istruzioni:

- CISC:
  - le istruzioni hanno dimensione variabile,
  - il campo codice operazione varia in lunghezza,
  - esiste un numero ampio di formati di istruzione: il codice di operazione identifica lo specifico formato delle istruzioni.
- RISC:
  - le istruzioni hanno dimensione fissa,
  - il campo del codice operazione ha uno spazio predefinito,
  - esiste un numero limitato di formati: il codice operazione identifica in modo univoco il formato dell'istruzione.



# Indirizzamento della memoria

- Le CPU moderne riferiscono gli indirizzi ai byte, però per ogni trasferimento CPU ←→ MEM è possibile trasferire anche 16, 32, 64, … bit alla volta se il bus lo consente.

# Indirizzamento: ordinamento della memoria

- Quando vengono trasferiti oggetti multipli di byte il loro ordinamento in memoria può differire in base alla convenzione adottata:

| Little Endian | | | | | Big Endian | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | 2 | 1 | 0 | 0 | 0 | 1 | 2 | 3 |
| 7 | 6 | 5 | 4 | 4 | 4 | 5 | 6 | 7 |

(esempio su 32 bit)

- L'ordinamento diventa importante quando si debba scambiare dati tra due macchine che adottano convenzioni diverse.



# Indirizzamento: allineamento della memoria

Le parole di 32 bit possono essere allocate in memoria:

- ad indirizzi multipli di 4 byte,
  - → è richiesto un solo accesso alla memoria per leggere una parola;
- ad indirizzi qualsiasi
  - → sono richiesti due accessi alla memoria per leggere una parola;

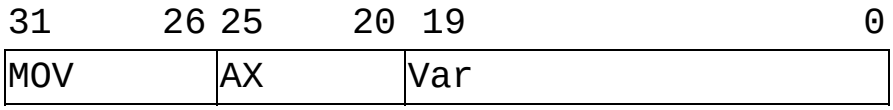
Alcune architetture (RISC) non permettono accessi *non allineati* alla memoria

# Indirizzamento dei dati: problema

Si consideri:

`MOV AX, Var  ; AX ← M[Var]`

e il formato dell'istruzione corrispondente:




come si nota l'indirizzo `Var` ha a disposizione solo 20 bit per formare l'indirizzo.

→ Per poter indirizzare tutta la memoria (se più grande della dimensione rappresentabile su 20 bit) è necessario disporre di metodi alternativi per l'indirizzamento.

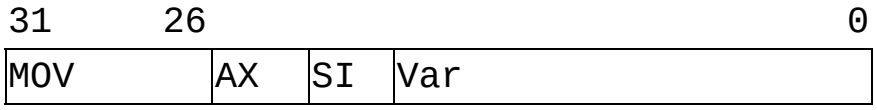


# Indirizzamento dei dati: soluzione per il calcolo dell'indirizzo effettivo

Si consideri:

`MOV AX, Var[SI]  ; AX ← M[Var + SI]`

avente il seguente formato dell'istruzione:




In questo caso l'indirizzo effettivo è dato dalla somma del contento del campo Var con il contenuto del registro BX.

→ Questo è un indirizzamento che fa uso di un scostamento o di un registro indice.

# Memoria lineare e segmentata

Il calcolo dell'indirizzo effettivo dipende al modello di memoria adottato. I due modelli principali sono:

- **memoria lineare**
  - Locazioni da 0 a M-1: in questo caso Var si calcola rispetto all'indirizzo 0 – il *loader* si occupa di ricalcolare gli indirizzi della memoria.
- **memoria segmentata**
  - La memoria è divisa in segmenti caratterizzati da posizioni e dimensioni non fisse (anche sovrapposti): segmenti diversi possono essere assegnati rispettivamente a codice, stack e dati.
  - I segmenti sono contenuti all'interno di particolari registri selettore (es. CS, SS, DS, ES).
  - L'indirizzo fisico si ottiene sommando l'indirizzo effettivo al registro selettore.



# Riepilogo indirizzamento dei dati

- Indirizzamento diretto
  ```
  MOV AX, Var           ; AX←M[Var]
  ```
- Indirizzamento relativo ai registri
  ```
  MOV AX, Var[SI]       ; AX←M[Var+SI]
  ```
- Indirizzamento indiretto rispetto ai registri
  ```
  MOV AX, [BX]          ; AX←M[BX]
  ```
- Indirizzamento relativo ai registri scalato con indice
  ```
  MOV AX, Var[SI][BX]   ; AX←M[Var+SI+BX]
  ```
- Indirizzamento immediato
  ```
  MOV AX, 1234          ; AX←1234
  ```
- Indirizzamento dei registri
  ```
  MOV AX,BX             ; AX←BX
  ```
- Indirizzamento delle porte di I/O
  ```
  IN AL,PORTA_IN        ; AL←Porta in Ingresso
  ```