Corso di Architettura degli Elaboratori e Laboratorio (M-Z)

Insieme di istruzioni macchina

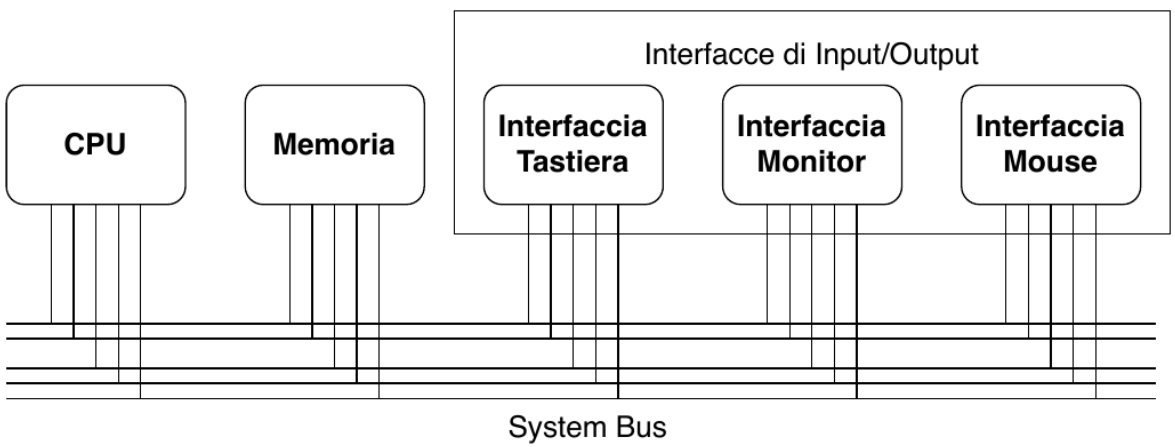
Nino Cauli

UNIVERSITÀ
degli STUDI
di CATANIA

Dipartimento di Matematica e Informatica

Architettura base di un calcolatore elettronico

- **CPU**: esegue istruzioni elementari
- **MEMORIA**: contiene il programma (sequenza di istruzioni elementari) che la CPU deve eseguire e i dati necessari
- **INTERFACCE DI INPUT/OUTPUT**: circuiti elettronici che permettono di connettere la CPU al mondo esterno
- **BUS DI SISTEMA**: insieme di collegamenti elettrici che interconnettono I vari componenti di un calcolatore

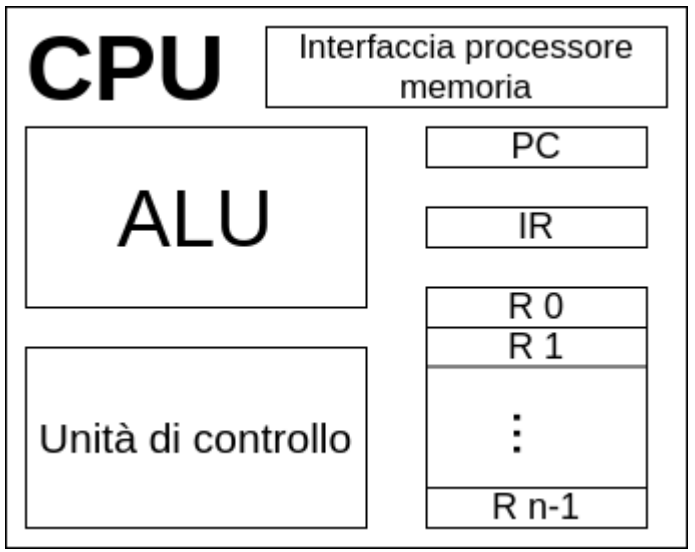


Come funziona?

- Il calcolatore elettronico esegue **SEQUENZIALMENTE** una serie di **ISTRUZIONI**
- Le istruzioni definiscono delle operazioni da eseguire e sono raggruppate in **PROGRAMMI**
- Spesso le operazioni devono essere eseguite su dei **DATI**
- L'utente può interagire con il calcolatore tramite le **INTERFACCE DI I/O** (**PERIFERICHE**)

Processore (CPU)

- È un **CIRCUITO ELETTRONICO INTEGRATO** (chip) con il ruolo di **CERVELLO** del calcolatore
- Capace di caricare ed eseguire le **ISTRUZIONI ELEMENTARI** necessarie per eseguire i **PROGRAMMI**
- Esempi di istruzioni elementari: operazioni aritmetiche, operazioni logiche, confronti, salti incondizionati e condizionati.



Memoria

- Le unità memoria sono usate per immagazzinare informazione necessaria per eseguire i programmi

- Sono circuiti elettronici in grado di preservare l'informazione che può essere costituita da:
  - **ISTRUZIONI**, eseguite dalla CPU
  - **DATI**, utilizzati dalle istruzioni eseguite

- La memoria si può dividere in **MEMORIA CENTRALE** e **MEMORIA DI MASSA**

Esecuzione di un’istruzione

Passi operativi elementari per ciascuna istruzione:

- **PRELIEVO**: prelievo della prossima istruzione dalla memoria (scrivere la prossima istruzione nel registro di istruzione IR)
- **DECODIFICA**: decodifica dell’istruzione (quale operazione bisogna eseguire? Dove si trovano i dati da usare?)
- **ESECUZIONE**: esecuzione dell’istruzione (leggere o scrivere un dato in memoria, eseguire operazioni matematiche e logiche sui registri)

Rappresentazione di istruzioni e dati

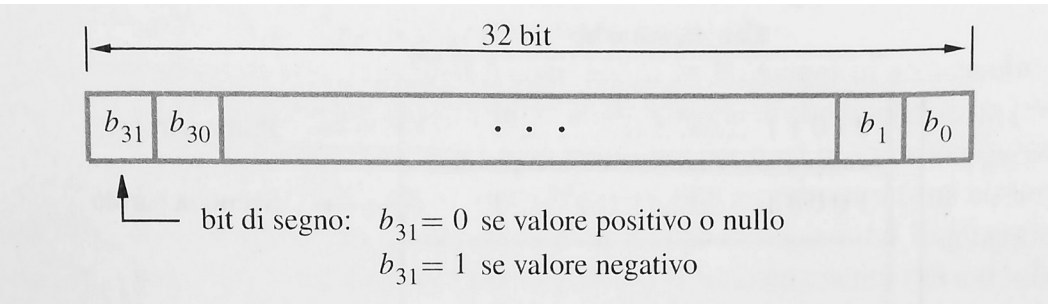
- Le istruzioni e i dati sono rappresentati da **SEQUENZE** di **CIFRE BINARIE** (**bit**)

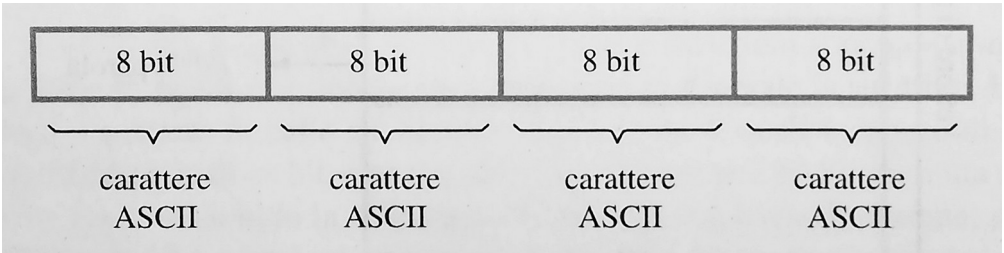
- Per convenzione una sequenza di 8 bit è detta **Byte**

- I byte vengono raggruppati in blocchi con un numero di elementi espresso con potenze di 2:
  - Kilobyte = KB = 2^{10} = 1024 ≈ 10^3
  - Megabyte = MB = 2^{20} = 1024 * 1024 ≈ 10^6
  - Gigabyte= GB = 2^{30} = 1024 * 1024 * 1024 ≈ 10^9
  - Terabyte = TB = 2^{40} = 1024 * 1024 * 1024 * 1024 ≈ 10^{12}

Parola di memoria (memory word)

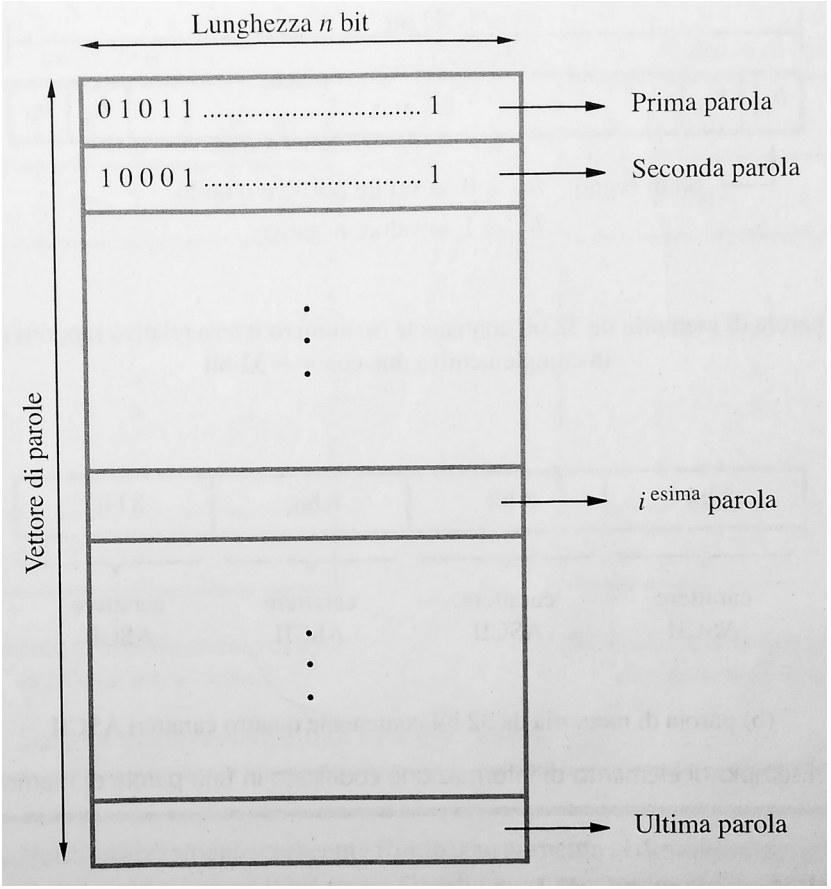
- Il calcolatore non lavora su singoli bit ma su gruppi di bit detti **PAROLE** di lunghezza da 8 a 64 bit (sempre potenze di 2)
- La dimensione delle parole dipende dall'architettura del calcolatore
- I dati posso occupare da un singolo byte a diverse parole
- Le istruzioni possono occupare una o più parole





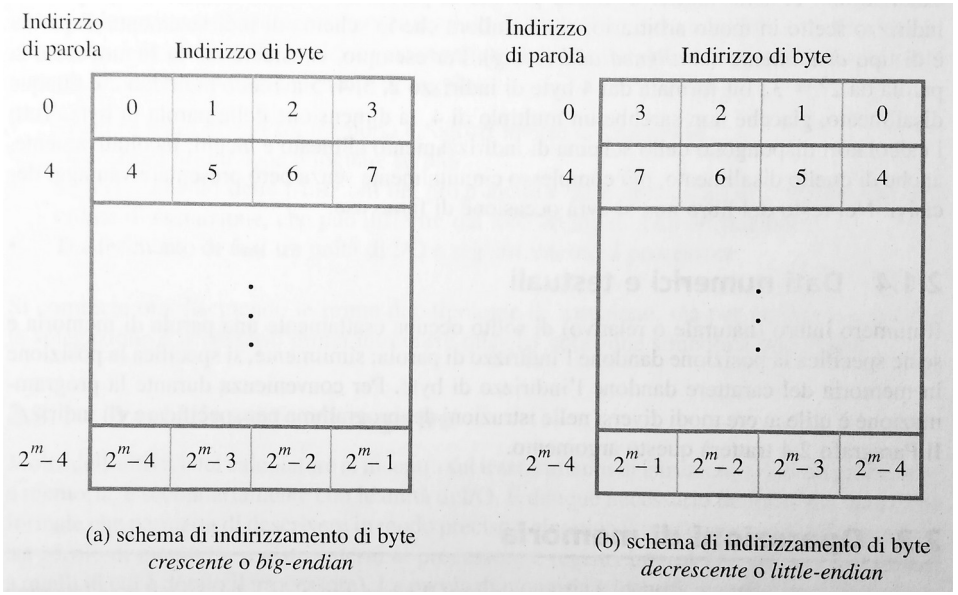
Organizzazione della memoria

- L’informazione è immagazzinata in memoria sotto forma di un vettore di parole (parole in successione)
- Ad ogni parola nel vettore è associato un indirizzo binario univoco
- Un numero binario di m bit può rappresentare 2^m indirizzi
- Parole consecutive sono associate ad indirizzi consecutivi
- L’insieme degli indirizzi associati a ciascuna parola di memoria è chiamato spazio di indirizzamento



Indirizzamento e ordinamento di byte

- Di norma l'unità minima di informazione indirizzabile in memoria è il byte
- Si assegnano indirizzi consecutivi ai byte contenuti in ciascuna parola
- Gli indirizzi delle parole saranno quindi multipli della loro lunghezza in byte
- Vi sono 2 schemi di indirizzamento di byte:
  - **Crescente (big-endian)**: indirizzo aumenta al diminuire del peso aritmetico del byte
  - **Decrescente (little-endian)**: indirizzo aumenta all'aumentare del peso aritmetico del byte



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

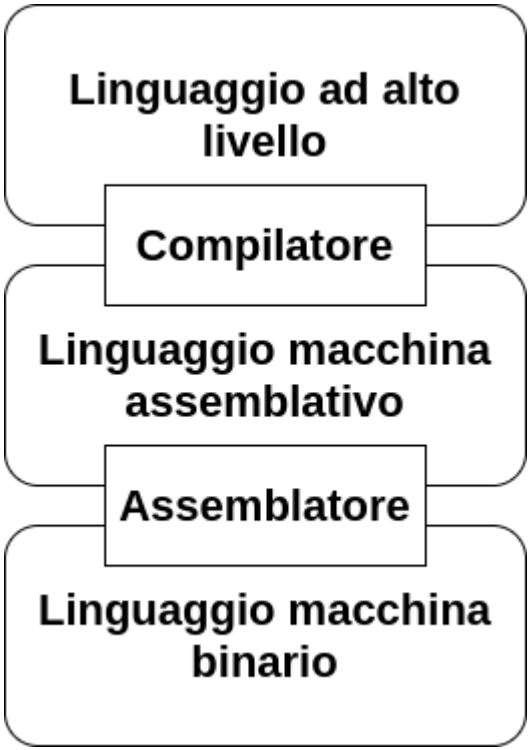
(b) schema di indirizzamento di byte *decrescente* o *little-endian*

Instruction set architecture ISA

- Il processore è in grado di eseguire un insieme di operazioni base chiamate istruzioni macchina
- L'insieme delle istruzioni eseguibili da un processore e le loro modalità d'uso è chiamato ISA (Instruction Set Architecture)
- Ogni processore commerciale ha il suo specifico ISA
- Il linguaggio macchina permette di definire le istruzioni attraverso un alfabeto binario {0, 1}
- Il linguaggio assemblativo è una rappresentazione simbolica leggibile del linguaggio macchina

Come si programma?

- Il programmatore scrive i programmi in **LINGUAGGIO ASSEMBLATIVO** (**ASSEMBLY**)
- Il programma assemblativo viene tradotto in sequenze binarie dall'**ASSEMBLATORE**
- Linguaggi ad alto livello (C, C++, etc.) ancora più espressivi
- Il **COMPILATORE** traduce il codice ad alto livello in codice assemblativo



Insiemi di istruzioni RISC e CISC

Esistono due approcci nella progettazione dell'insieme di istruzioni dei calcolatori:

Reduced Instruction Set Computer (RISC):

- Insieme di istruzioni base ridotto
- Ogni istruzione occupa una sola parola di memoria
- Gli operandi delle istruzioni aritmetiche e logiche devono trovarsi nei registri del processore
- Prestazioni elevate grazie ad un'elaborazione a stadi (pipeline)

Complex Instruction Set Computer (CISC):

- Insieme di istruzioni base complesse
- Ogni istruzione può occupare più di una parola di memoria
- Gli operandi delle istruzioni aritmetiche e logiche possono trovarsi in memoria

Istruzioni macchina di base

- I programmi eseguiti da un calcolatore sono composti da una sequenza di istruzioni base (addizione, confronto, caricamento di dati, ecc.)

- L'insieme di istruzioni riconosciute deve comprendere almeno queste quattro tipologie:
  - Trasferimento dati tra memoria e registri del processore
  - Operazioni aritmetiche e logiche sui dati
  - Operazioni di controllo dell'ordine di esecuzione delle istruzioni
  - Trasferimento dati tra unità di I/O e registri del processore

Linguaggio assemblativo RISC generico

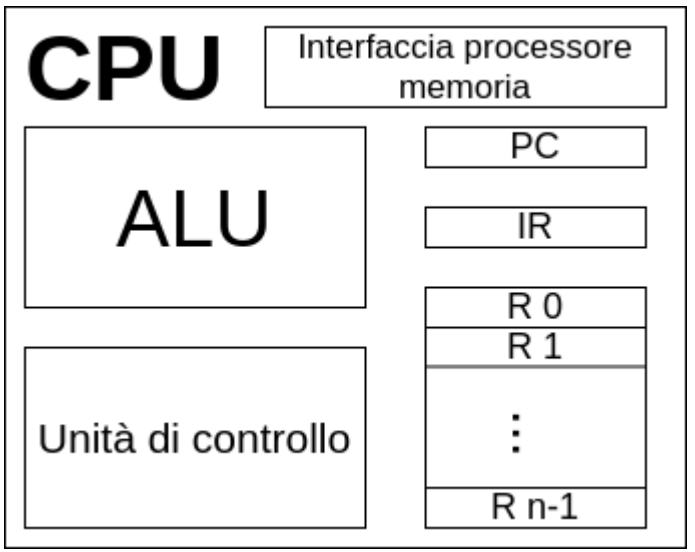
- Le diverse ISA dei processori commerciali posseggono linguaggi assemblativi con formalismi differenti (sebbene simili)
- Nella teoria di questo corso useremo un linguaggio assemblativo generico non appartenente a nessun processore commerciale
- Utile per capire i concetti base che possono essere applicati a qualsiasi architettura
- Verrà presentato un set di istruzioni base per programmare un processore nella pratica (non il set di istruzioni completo)

Registri e locazioni di memoria

- È necessario definire una notazione formale per riferirsi ai registri e alle locazioni di memoria nel linguaggio assemblativo generico

- I registri sono identificati attraverso il loro nome:
  - Registri generici del processore: R0, R1, …, Rn
  - Registri speciali del processore: PC, IR, ecc.
  - Registri di I/O: INGRESSO_DATO, USCITA_DATO, ecc.

- Le locazioni di memoria sono identificate attraverso il loro indirizzo in forma:
  - Di costante numerica
  - Di costante simbolica dichiarata in precedenza: VAR1, IND, CICLO, ecc.



Istruzioni base per accesso alla memoria

Load *destinazione sorgente*

- Istruzione usata per caricare un dato dalla memoria ad un registro del processore
- Il campo destinazione è il nome di un registro del processore
- Il campo sorgente è una locazione di memoria
- La locazione di memoria può essere indicata in vari modi a seconda del modi di indirizzamento usato

Store *sorgente destinazione*

- Istruzione usata per salvare in memoria un dato presente in un registro del processore
- Il campo sorgente è il nome di un registro del processore
- Il campo destinazione è una locazione di memoria
- La locazione di memoria può essere indicata in vari modi a seconda del modi di indirizzamento usato

Istruzioni base per somma e sottrazione

Add *destinazione sorgente1 sorgente2*

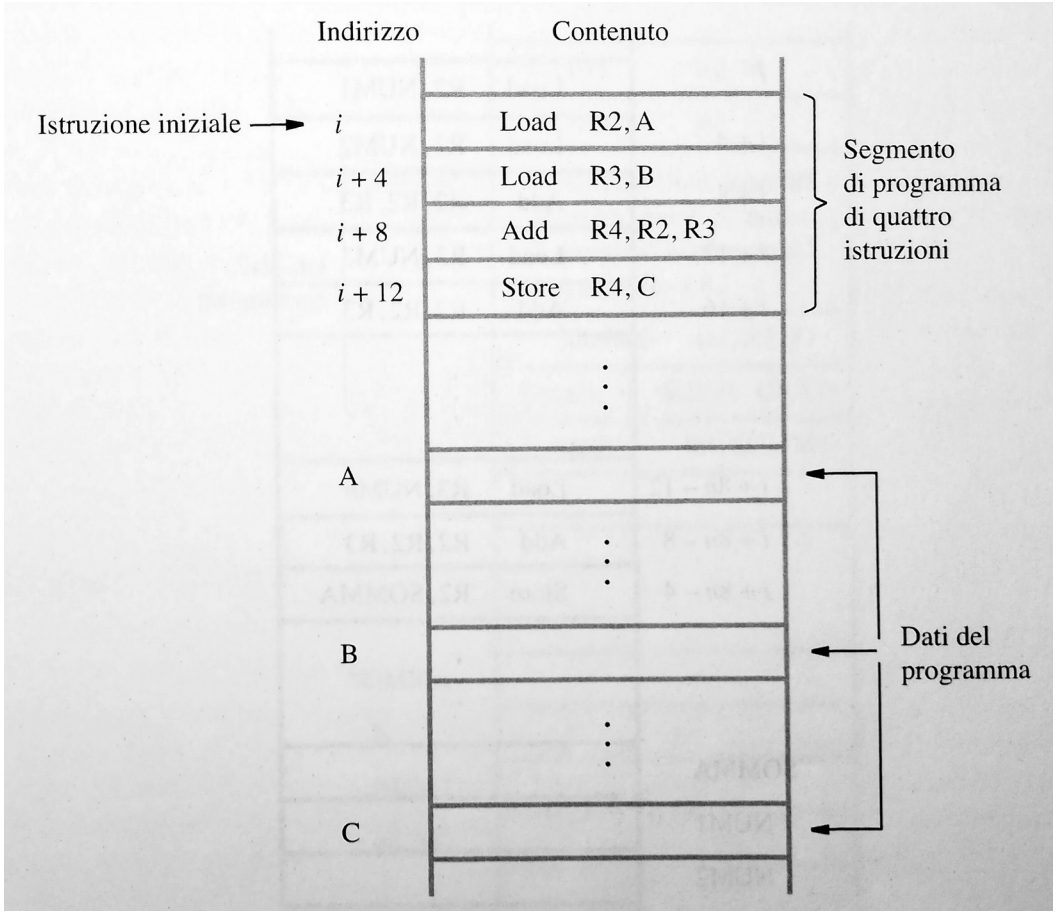
- Istruzione usata per sommare il contenuto di due registri
- Il campo destinazione è il nome di un registro del processore su cui scrivere la somma
- I campi sorgente1 e sorgente2 rappresentano i numeri da sommare
- Gli addendi possono essere espressi come nomi di registri o direttamente come valore

Subtract *destinazione sorgente1 sorgente2*

- Istruzione usata per sottrarre il contenuto di due registri
- Il campo destinazione è il nome di un registro del processore su cui scrivere la differenza
- I campi sorgente1 e sorgente2 rappresentano i numeri da sotrarre
- Gli operandi possono essere espressi come nomi di registri o direttamente come valore

Esempio di programma di somma

- Esempio di programma che somma due valori presenti in memoria e ne salva il risultato
- Programma composto da 4 istruzioni (2 Load, 1 Add e 1 Store)
- Le quattro istruzioni sono memorizzate in parole di memoria consecutive
- Istruzioni lette sequenzialmente
- PC contiene l'indirizzo della prossima istruzione da eseguire e IR contine l'istruzione in esecuzione



Modi di indirizzamento

- Nel linguaggio assemblativo, gli operandi e il risultato delle istruzioni possono essere espressi in modi diversi
- I metodi con cui specificare operandi e risultato vengono chiamati modi di indirizzamento
- I modi di indirizzamento base di un'architettura RISC sono:
  - **Modo immediato**
  - **Modo di registro**
  - **Modo assoluto (diretto)**
  - **Indiretto da registro**
  - **Con indice e spiazzamento**
  - **Con base e indice**

Modi di registro e assoluto

- I modi di indirizzamento visti fino ad ora sono:
  - **Modo di registro:** Il nome (= indirizzo) di un registro di processore contenente l'operando o il risultato è dato nell'istruzione
  - **Modo assoluto (diretto):** L'indirizzo di una parola di memoria contenente l'operando o il risultato è dato nell'istruzione

- Nei processori RISC c'è un limite al numero di bit per un indirizzo assoluto (un'istruzione = una parola)

- Per processori a 32 bit = indirizzo assoluto 16 bit

Load R2, NUM1

Modo immediato

- Per usare una costante numerica come operando si ricorre al modo di indirizzamento immediato
- **Modo immediato:** L'operando è dato esplicitamente nell'istruzione
- Si precede la costante dal simbolo cancelletto: **#valore**
- Esempio in cui si aggiunge il valore 200 al contenuto di R6 e si pone il risultato in R4:

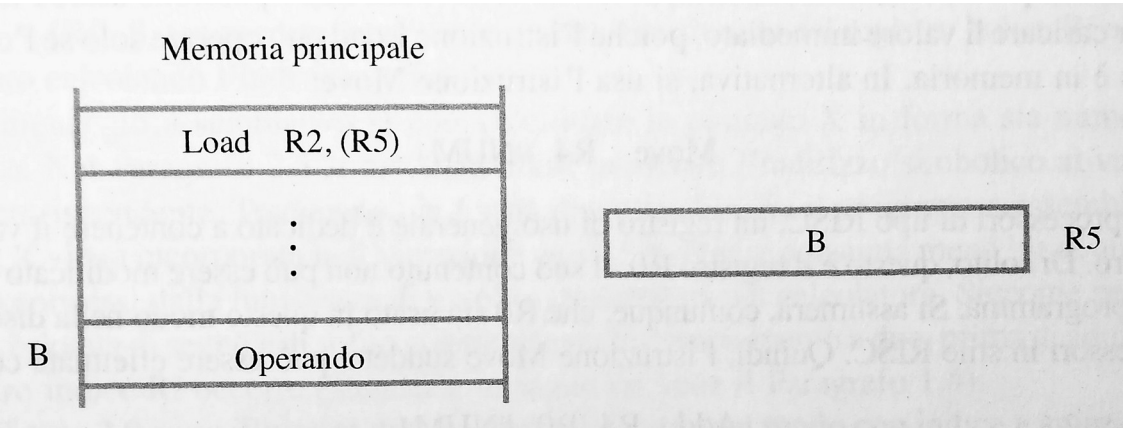
Add R4, R6, #200

Modo indiretto da registro

- **Modo indiretto:** Il nome di un registro di processore contenente l'INDIRIZZO di memoria dell'operando o del risultato è dato nell'istruzione

- Viene rappresentato con il nome del registro tra parentesi tonde **(·)**

- Usato in casi in cui si voglia riutilizzare una stessa istruzione in memoria più volte cambiando gli operandi



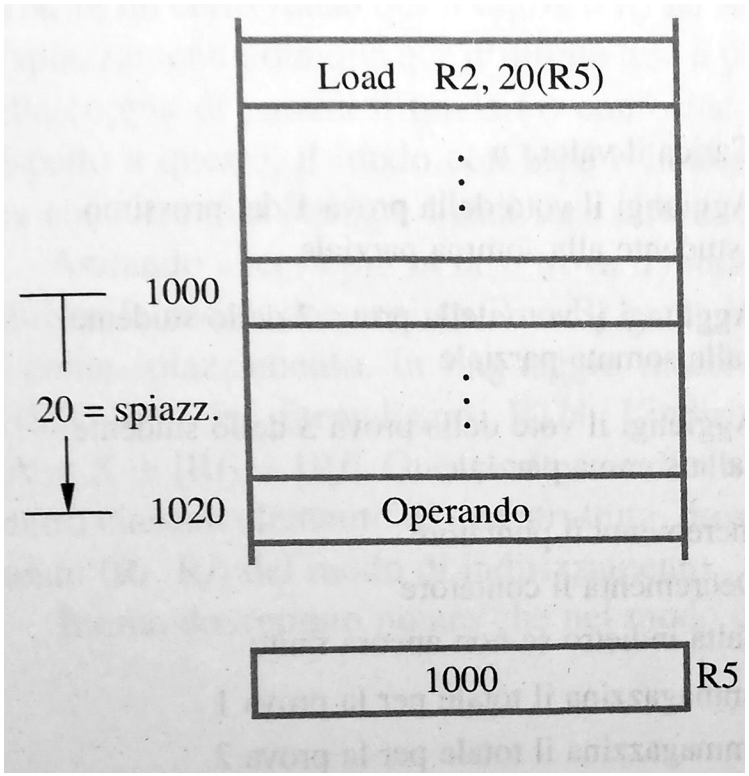
Modo con indice e spiazzamento

- **Modo con indice e spiazzamento:** L'indirizzo effettivo di operando o risultato è ottenuto addizionando un valore costante (spiazzamento) al contenuto di un registro (indirizzo)

- Per indicare indice e spiazzamento si usa la scrittura X(Ri), dove X è lo spiazzamento e Ri è il nome del registro contenente l'indirizzo

- Utile nel gestire vettori o liste

- Esistono versioni più complesse come il **modo con base e indice** dove l'indirizzo effettivo è ottenuto sommando il contenuto di due registri, denotato così: (Ri, Rj)



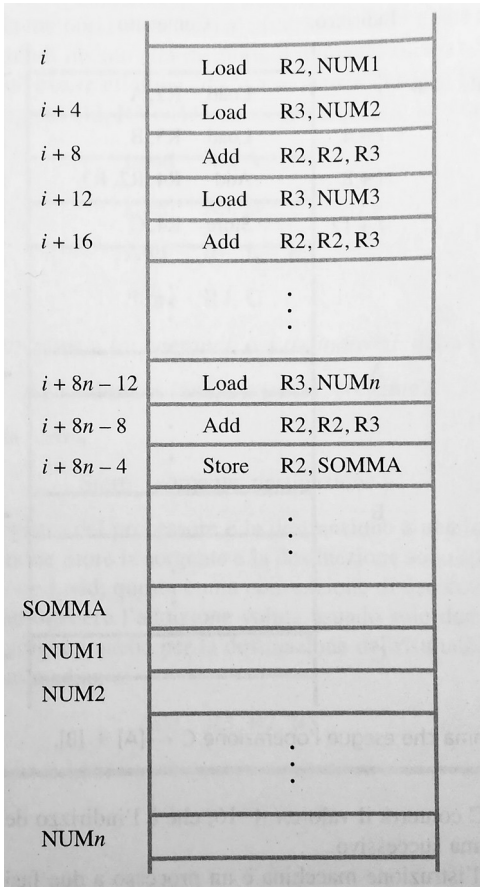
Istruzione di salto condizionato (Branch_if)

Branch_if_condizione *destinazione_salto*

- Istruzione usata per saltare all'esecuzione di un'istruzione specifica nel caso la condizione di salto sia vera
- La condizione di salto può essere tra valori contenuti nei registri (espressi tra quadre: [Ri]) o valori espressi esplicitamente
- La destinazione del salto è espressa come locazione di memoria contenente l'istruzione da eseguire nel caso la condizione sia vera
- Esempio di salto all'istruzione CICLO nel caso il contenuto di R2 sia maggiore di 0:

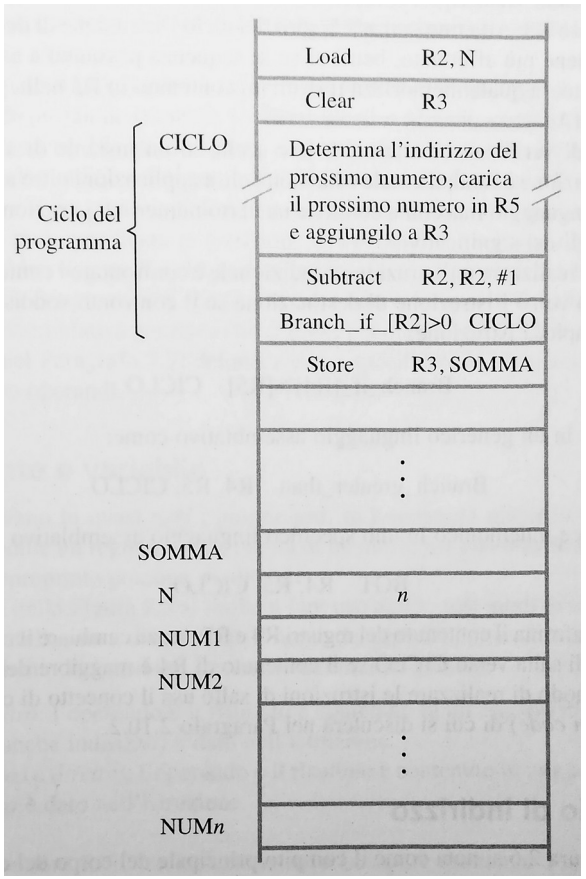
Branch_if_[R2]>0 CICLO

Esempio somma di n numeri



Esempio sequenziale (poco efficiente)

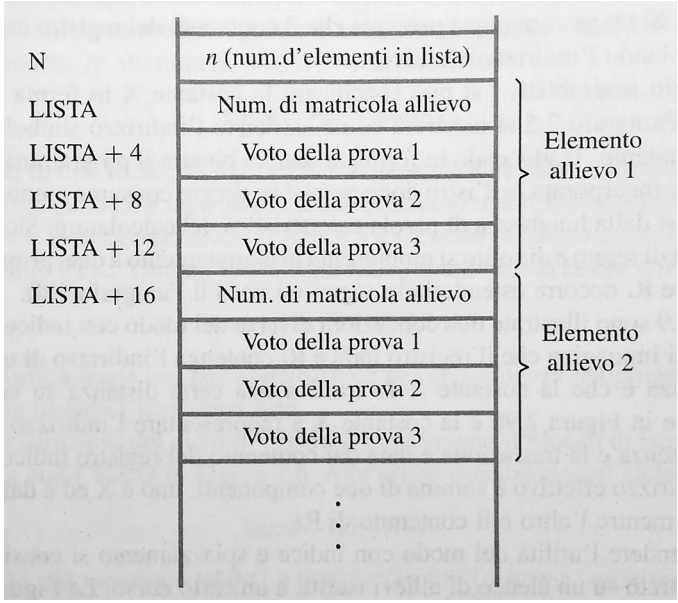
Esempio con salto (più efficiente)



Esempio somma di n numeri con salto

	Load	R2,N	Carica la dimensione della lista
	Clear	R3	Inizializza la somma a 0
	Move	R4,#NUM1	Carica l'indirizzo del primo numero
CICLO:	Load	R5, (R4)	Preleva il prossimo numero
	Add	R3, R3, R5	Aggiungi questo numero alla somma
	Add	R4, R4, #4	Incrementa il puntatore alla lista
	Subtract	R2, R2, #1	Decrementa il contatore
	Branch_if_[R2]>0	CICLO	Salta indietro se non ancora finito
	Store	R3, SOMMA	Immagazzina la somma finale

Esempio voti studenti




```
        Move            R2, #LISTA      Carica l'indirizzo LISTA
        Clear           R3
        Clear           R4
        Clear           R5
        Load            R6, N           Carica il valore n
CICLO:  Load            R7, 4(R2)       Aggiungi il voto della prova 1 del prossimo
        Add             R3, R3, R7        studente alla somma parziale
        Load            R7, 8(R2)       Aggiungi il voto della prova 2 dello studente
        Add             R4, R4, R7        alla somma parziale
        Load            R7, 12(R2)      Aggiungi il voto della prova 3 dello studente
        Add             R5, R5, R7        alla somma parziale
        Add             R2, R2, #16     Incrementa il puntatore
        Subtract        R6, R6, #1      Decrementa il contatore
        Branch_if_[R6]>0 CICLO          Salta indietro se non ancora finito
        Store           R3, SOMMA1      Immagazzina il totale per la prova 1
        Store           R4, SOMMA2      Immagazzina il totale per la prova 2
        Store           R5, SOMMA3      Immagazzina il totale per la prova 3
```