Lezione di informatica del 31 marzo 2008

(appunti prelevati sulla rete) - Paolo Latella

ARCHITETTURA DI UN CALCOLATORE ELETTRONICO

Per architettura di un calcolatore elettronico si intende l'insieme delle principali unità funzionali di un calcolatore ed il modo in cui queste interagiscono.

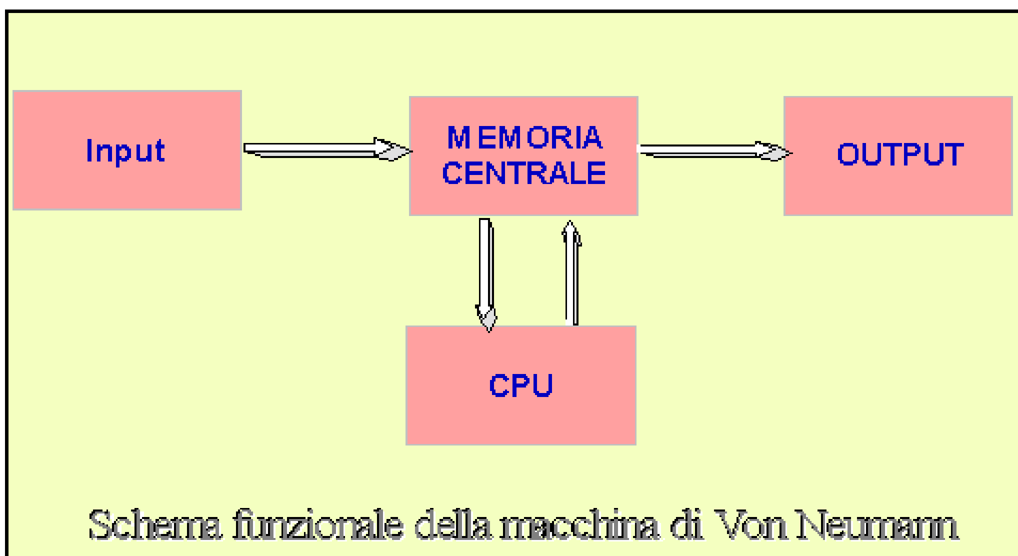
Funzioni di base di un calcolatore:

- memorizzazione dei dati
- elaborazione dei dati
- trasferimento dei dati
- controllo

L'architettura della maggior parte dei calcolatori elettronici è organizzata secondo il modello della **Macchina di Von Neumann.**

La Macchina di Von Neumann è costituita da quattro elementi funzionali fondamentali:

- il processore o unità centrale di elaborazione (CPU Central Processing Unit),
- la memoria centrale,
- i dispositivi di ingresso e uscita
- il bus di sistema.



Schema funzionale della macchina di Von Neumann

La CPU

L'unità centrale di elaborazione (CPU) è la parte del sistema che contiene gli elementi circuitali necessari al funzionamento dell'elaboratore.
Questa esegue i programmi che risiedono nella memoria centrale, prelevando, decodificando ed eseguendo le istruzioni in essi contenute e coordinando il trasferimento dei dati tra le varie unità funzionali.

La CPU si compone di:

- **una unità di controllo (CU Control Unit),** che ha lo scopo di interpretare e attivare le risorse necessarie alla esecuzione delle istruzioni,
- **una unità aritmetico-logica (ALU Arithmetic and Logic Unit)** in cui vengono effettuati i calcoli aritmetici e logici presenti nelle istruzioni (aritmetiche/logiche) del programma e
- **alcuni dispositivi di memoria detti *registri*** .

REGISTRO

Dal punto di vista tecnologico un registro è un insieme di n elementi fisici bistabili, detti bit. Poiché ciascun elemento ha due configurazioni stabili possibili, a cui per convenzione vengono associati i simboli 0 e 1, si ha che un registro formato da n bit è in grado di assumere 2^n configurazioni di stato diverse.
Lo stato del registro, ossia la configurazione dei suoi bit, rappresenta l'informazione che vi è memorizzata e tale informazione viene conservata fino a quando non la si altera.
Un registro quindi ha una capacità di memorizzare informazioni che è funzione del numero degli elementi di cui si compone.

I registri fondamentali presenti nella CPU sono:

- il registro degli indirizzi di memoria (***MAR Memory Address Register***), indica l'indirizzo della locazione di memoria che si vuole selezionare;

- il registro dei dati di memoria (***MDR Memory Data Register***), contiene il dato proveniente dalla locazione di memoria selezionata o il dato che si vuole memorizzare nella locazione di memoria selezionata;

- il contatore di programma (***PC Program Counter***) ha la funzione di guidare il flusso della esecuzione di un programma, infatti il suo contenuto indica l'indirizzo della prossima istruzione da eseguire;

- il registro della istruzione corrente (***IR Instruction Register***) che contiene l'istruzione da decodificare e eseguire.

- il registro delle interruzioni (***INTR Interrupt Register***) che contiene informazioni sullo stato di funzionamento delle periferiche (la descrizione dell'uso di questo registro sarà ripresa durante lo studio dei sistemi operativi)

L'unità di controllo

La CU ha il compito di sovrintendere a tutte le attività del calcolatore, imponendo la corretta sequenzializzazione delle operazioni elementari che devono essere svolte nell'esecuzione del programma.

A tale scopo preleva dalla memoria centrale una alla volta le istruzioni che compongono il programma, le decodifica (tramite il decodificatore di istruzioni (ID)) e le esegue inviando gli opportuni segnali di controllo agli organi della CPU che ne attuano l'esecuzione.

ALU

L'unità aritmetico-logica (ALU) è costituita da:

- dispositivi circuitali che consentono di eseguire le operazioni aritmetiche somma, sottrazione, prodotto, divisione(ADD, SUB, MUL, DIV) o logiche (AND, OR, NOT) sugli operandi memorizzati nei registri interni all'ALU .
- alcuni registri interni.

I principali registri interni all'ALU sono:

- il **registro accumulatore** (**A**), dove è memorizzato uno degli operandi coinvolti nell'operazione aritmetica o logica e dove rimane memorizzato il risultato di tale operazione;
- il registro operando(**OP**) dove può essere memorizzato un altro operando o dato temporaneo coinvolto nell'operazione aritmetica o logica;
- il registro di stato (**PSW** - Processor Status Word) i cui bit forniscono informazioni relative all'esito dell'ultima operazione aritmetico-logica eseguita.
  - il bit di carry (CF-carry flag) quando vale 1, indica la presenza di un riporto (carry), nell'istruzione di somma tra naturali indica che il risultato dell'operazione non è rappresentabile;
  - il bit zero (ZF-zero flag) quando vale 1 indica che l'ultima operazione eseguita ha prodotto zero (coiè tutti i bit nell'accumulatore valgono 0);
  - il bit di overflow (OF-overflow flag) quando vale 1 indica il verificarsi di overflow;
  - il bit del segno (SF-sign flag) quando vale 1 indica che l'ultima operazione eseguita ha prodotto un risultato con il bit più significativo pari a 1 ovvero nel caso di interi si tratta di un numero negativo;

  Questi flag sono interpretati dalla CU che è in grado di intraprendere azioni differenziate a seconda dei risultati dei calcoli effettuati nell'ALU.

La Memoria Centrale

Per **memoria si intende un dispositivo in grado di immettere, conservare ed estrarre informazioni.**
La **memoria centrale** è la memoria interna al calcolatore, **direttamente accessibile dalla CPU,** essa contiene i programmi e i dati necessari all'esecuzione dei programmi.
In un sistema di elaborazione si trovano sempre due tipi di memoria:

- la **memoria centrale**, interna al calcolatore, **direttamente accessibile dalla CPU** realizzata da *componenti a semiconduttore*;

- la **memoria secondaria o di massa**, **esterna al calcolatore** realizzata da *componenti magnetici o ottici*(dischi, nastri).

Dal punto di vista logico la **memoria centrale** è:

- **un insieme finito di locazioni (celle o registri) di uguali dimensioni;**

- **ogni locazione di memoria consta di n elementi circuitali bistabili, detti bit,** ciascuno dei quali può rappresentare una informazione binaria, ovvero una informazione che può assumere solo i valori 0 e 1.
- **Ogni locazione è caratterizzata da un indirizzo e dal contenuto.**
- Si chiama **indirizzo di una locazione** la **posizione che questa occupa nella memoria rispetto alla prima locazione che ha indirizzo zero.**
- Il **contenuto di una locazione** ovvero l'informazione in essa registrata si chiama ***parola*** di memoria.

Mentre il bit rappresenta l'unità elementare di informazione, ***la locazione di memoria è*** la più piccola quantità di memoria accessibile ovvero che possiede un indirizzo ***pertanto rappresenta l'unità di informazione*** scambiata tra i vari elementi funzionali di cui si compone l'architettura ed il numero di bit di cui si compone ne determina il parallelismo.

Caratteristiche della memoria centrale

La memoria centrale è caratterizzata dalla dimensione della parola, la lunghezza di una parola di memoria può essere 8,16,32,64 bit, **e dalla capacità ovvero dal numero totale di locazioni di memoria.**

Poiché la dimensione della parola di memoria può variare da calcolatore a calcolatore, ma comunque è un multiplo del byte, la capacità della memoria si misura in byte o in multipli del byte.

Sono multipli del byte: il Kilo byte rappresentato dal simbolo KB, il Mega byte (MB), il Giga byte (GB), il Tera byte (TB).
I termini Kilo, Mega, Giga e Tera vengono associati alle seguenti potenze di 2:

1KB = 2^{10} = 1024 ≈ 10^3 (mille) byte
1MB = 2^{20} = 1024 K ≈ 10^6 (un milione) byte
1GB = 2^{30} = 1024 M ≈ 10^9 (un miliardo) byte
1TB = 2^{40} = 1024 G ≈ 10^{12} (mille miliardi) byte

Ciascuna locazione di memoria può essere selezionata specificando, nel registro degli indirizzi (MAR Memory Address Register), il suo indirizzo ossia la sua posizione rispetto alla prima cella di memoria, a cui viene attribuita per convenzione la posizione (ovvero l'indirizzo) zero.

L'indirizzamento di una locazione di memoria consiste nel selezionare elettricamente la locazione relativa all'indirizzo specificato.

Se il (MAR) registro degli indirizzi ha k bit può indirizzare 2^k celle di memoria i cui indirizzi variano da 0 a 2^k-1.
In particolare con un MAR di 10 bit si possono indirizzare 2^{10} ovvero 1024 locazioni di memoria.

Alla memoria centrale si accede per effettuare operazioni di lettura o scrittura.

La lettura di una locazione di memoria consiste nel trasferimento fisico dei byte che costituiscono la locazione dalla memoria alla unità centrale di processamento.

Una operazione di lettura consiste nei seguenti passi:

- si scrive sul MAR l'indirizzo della locazione da leggere,
- questo poi viene trasferito al bus degli indirizzi che trasporta l'indirizzo in memoria,
- la quale trascorso il tempo d'accesso (in genere dell'ordine dei nanosecondi (nano=10^{-9})) scrive a sua volta sul bus dei dati il contenuto della locazione di memoria selezionata,
- che successivamente viene inserita o caricata (*load*) nel regitro dei dati (MDR Memory Data Register) restando così disponibile alla CPU.

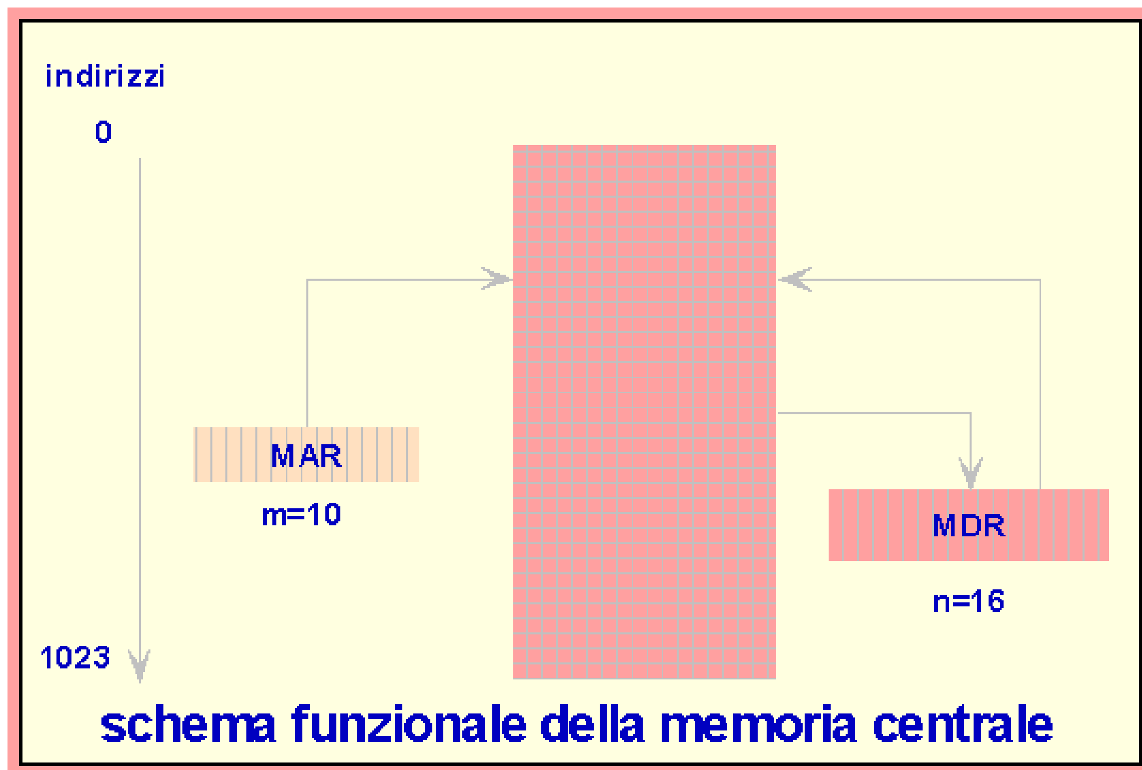
La scrittura in una locazione di memoria consiste nel trasferimento fisico del contenuto del registro dei dati (MDR) nella cella di memoria selezionata tramite il registro MAR.

Una operazione di scrittura consiste nei seguenti passi:

- si scrive sul registro MDR il dato da inserire o immagazzinare (*store*),
- si scrive sul MAR l'indirizzo della locazione da ricoprire (con il dato contenuto nel registro MDR),

- questo poi viene trasferito al bus che trasporta l'indirizzo in memoria,
- la quale trascorso il tempo di accesso scrive il dato che arriva dal bus dati nella locazione di memoria selezionata.

Schema di funzionamento della memoria centrale



La memoria centrale a cui ci siamo fino ad ora riferiti viene anche detta memoria RAM (Random Access Memory). Con questo termine si vuole indicare la possibilità di accedere in modo casuale a una qualsiasi locazione di memoria per effettuare operazioni di lettura o scrittura.

In effetti nei calcolatori reali una porzione di memoria centrale, è realizzata con diversa tecnologia e viene identificata con il termine ROM (Read Only Memory) in quanto è riservata solo per operazioni di lettura. Le ROM vengono usate dai costruttori per memorizzare in modo permanente informazioni necessarie per l'avviamento del sistema.

Dispositivi di Ingresso/Uscita e Unità periferiche

I dispositivi o interfacce di input/output permettono la comunicazione e quindi il trasferimento dei dati tra calcolatore e unità periferiche e viceversa.

Le unità periferiche vere e proprie, anche se sono componenti essenziali di un sistema informatico, non fanno parte della macchina di Von Neumann.

Le unità periferiche principali sono ***tastiera,mouse***, ***video, stampante***, i lettori magnetici e ottici, i plotter, le unità di memoria secondaria o di massa.

Le **interfacce** di I/O, diverse a seconda del tipo di periferica a cui devono essere collegate (unità di memoria di massa, stampanti, video ad alta risoluzione), sono dotate di:

- un registro dati RDP (per scambiare dati con la periferica),
- un registro comandi (per impartire comandi alla periferica) ed
- un registro di stato (per conoscere lo stato della periferica).

Il registro dati viene collegato al bus dati, il registro comandi al bus controlli ed il registro di stato interagisce con il registro delle interruzioni della CPU.

Il Bus di sistema

Il bus di sistema è il mezzo che, sotto il diretto controllo della CPU, collega le varie unità funzionali della macchina di Von Neumann.

Tramite il bus è possibile collegare **solo due unità funzionali alla volta**, una che **trasmette dati e una che riceve.**

E' la CPU che tramite i suoi organi assegna il bus ad uno specifico collegamento.

Nelle implementazioni concrete il bus di sistema è costituito da tre parti distinte:

- una monodirezionale dal processore alla memoria detta ***bus degli indirizzi***,
- una bidirezionale dal processore alla memoria e viceversa detta ***bus dei dati*** ed
- una monodirezionale dal processore alle altre unità funzionali detto ***bus dei comandi***.

Il bus degli indirizzi, connesso al MAR, ha tanti conduttori quanti sono i bit di un indirizzo (pari quindi alla dimensione del MAR). Questo numero in genere è il logaritmo in base due dello spazio di indirizzamento, ovvero del numero di locazioni di memoria complessivamente indirizzabili.

Il bus dei dati, connesso al registro MDR ha un numero di conduttori pari alla dimensione di una locazione di memoria, lo stesso numero quindi corrisponde anche alla lunghezza del registro MDR.

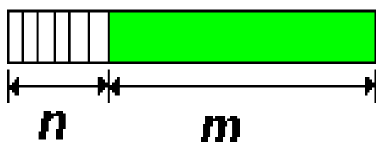
Il Linguaggio macchina

Per linguaggio macchina di un calcolatore si intende il linguaggio interpretabile direttamente dalla unità di controllo di un calcolatore.
E' rappresentato in codice binario ed è strettamente legato alla struttura logico-circuitale del calcolatore e alle operazioni elementari che questo sa eseguire.

I linguaggi macchina si distinguono soprattutto per:

- la struttura delle istruzioni e
- per il repertorio o set di istruzioni eseguibili.

In generale la struttura delle istruzioni di un linguaggio macchina consiste di due parti:



- una di *n* bit adibita alla rappresentazione del codice operativo
- una di *m* bit adibita alla rappresentazione di uno o più operandi.

Il codice operativo specifica l'azione da compiere, gli operandi specificano il dato oppure in base a varie modalità le locazioni di memoria in cui questo risiede.

Mentre il codice operativo è sempre presente gli operandi possono mancare.

Il *repertorio di istruzioni* che la macchina può eseguire comprende operazioni molto elementari, quali ad esempio il trasferimento di dati da un indirizzo di memoria a un registro della CPU e viceversa o operazioni aritmetiche/logiche su dati contenuti in specifici registi o operazioni di salto.

Le istruzioni di un qualsivoglia linguaggio macchina si possono raggruppare nelle seguenti classi:

- ***Istruzioni di interazione con la memoria centrale***
- ***Istruzioni di I/O***
- ***Istruzioni aritmetico-logiche***
- ***Istruzioni di controllo del flusso***

Nel caso in cui le istruzioni siano ad un solo operando e occupino complessivamente una locazione di memoria, dovrà verificarsi che se *m* è il numero di bit per la rappresentazione del codice operativo e *n* è il numero dei bit per identificare l'operando, la lunghezza *s* di ogni locazione di memoria risulta pari a *m+n.*

Ciò implica che la cardinalità del set di istruzioni del linguaggio dovrà essere minore o uguale di 2^n.

Inoltre il numero di locazioni di memoria indirizzabili, dove può risiedere l'operando, dovrà essere minore o uguale di 2m.

Per fissare le idee se supponiamo che le locazioni di memoria siano di 16 bit, il codice operativo di 4 bit e l'indirizzo dell'operando di 12 bit, allora avremo un repertorio di al più 2^4=16 istruzioni e potremo specificare indirizzi di memoria da 0 a 2^{12}-1 .

In realtà nelle macchine in genere vengono usati sistemi di gestione degli indirizzi più complessi di quelli descritti e che consentono tramite registri di due soli byte di rappresentare indirizzi di memoria centrale superiori a 2^{16}.

Programma

Un programma realizzato in linguaggio macchina è una sequenza di istruzioni, codificate secondo le regole del linguaggio.

Una volta che il programma è introdotto in macchina ciascuna istruzione è memorizzata in una determinata locazione della memoria centrale e può essere identificata dal corrispondente indirizzo.

Poiché la realizzazione di programmi scritti direttamente in linguaggio macchina è complicata, in quanto ogni istruzione è espressa in codice binario, è nata l'esigenza di usare versioni simboliche dei linguaggi macchina, detti **linguaggi assembly.**

Tali linguaggi, non potendo essere interpretati direttamente dai circuiti dell'unità di controllo richiedono la traduzione nella equivalente versione in linguaggio macchina.

Dal momento che comunque anche scrivere programmi in linguaggio assembly (o assembler) è molto noioso e complicato, poiché ogni istruzione svolge solo funzioni elementari, è nata l'esigenza di progettare strumenti più potenti e più vicini al linguaggio naturale. Tali linguaggi sono comunemente detti **linguaggi simbolici ad alto livello**.

Il funzionamento della Macchina di Von Neumann

Un *programma eseguibile* dalla macchina di Von Neumann consiste in una lista di istruzioni registrate in memoria centrale, che devono essere eseguite una alla volta secondo l'ordine specificato nel programma fino a quando non si incontra un’istruzione di controllo, la quale può alterare il flusso sequenziale stabilendo il numero d’ordine della successiva istruzione da eseguire.

L'unità centrale di elaborazione, che, come riportato nello schema funzionale, può accedere solo alle informazioni contenute nella memoria centrale, estrae le istruzioni del programma eseguibile a partire da quella il cui indirizzo si trova nel registro **(PC) contatore**

di programma , le decodifica e le esegue secondo **tre fasi** dette di **fetch, decode e execute,** fino a quando viene eseguita l'istruzione di halt.

Le fasi di elaborazione sono scandite da un segnale (di tensione a onda quadra da 0 a +5V generato da un oscillatore interno alla CPU, detto orologio di sistema) detto di *clock*, emesso appositamente per mantenere la sincronizzare tra le varie unità.
L'intervallo che intercorre tra due successivi impulsi è detto periodo di clock.
La frequenza del segnale di clock viene misurata in MHz (Mega Hertz, milioni di cicli al secondo).
Maggiore è la frequenza, maggiore è la velocità della CPU.
Una CPU da 200 MHz ha un periodo di clock di 5 ns ($1/(200*10^6)$s)

Fase di fetch

All'inizio della fase di fetch il contenuto del PC, che come detto, contiene l'indirizzo della prossima istruzione da eseguire, viene trasferito nel MAR e da lì sul bus degli indirizzi dando inizio al reperimento (fetch) e alla lettura della istruzione da eseguire.
Trascorso il tempo d'accesso in memoria la locazione di memoria selezionata, contenente l'istruzione da eseguire, viene depositata sul bus dati e da lì giunge sul registro MDR, e in fine nel registro delle Istruzioni IR (Instruction Register). Al termine della fase di fetch della istruzione **l'unità di controllo incrementa di uno il contenuto del PC,** per predisporsi ad eseguire l'istruzione successiva.

Fase di decode

Il registro istruzioni (IR) dato il formato delle istruzioni è logicamente diviso i due parti: la prima parte contiene il codice operativo e la seconda parte l'operando.
Pertanto inizia la fase di decodifica del codice operativo a carico dell'unità di controllo la quale a seconda della operazione decodificata provvederà all'esecuzione della istruzione stessa.

Fase di execute

L'esecuzione della istruzione può comportare nuovi accessi in memoria per il recupero degli operandi (fetch operandi), in questo caso, prima della esecuzione vera e propria della istruzione, viene eseguita la fase di fetch degli operandi.

Quando tutto ciò che comporta l'istruzione è caricato nei registri opportuni del processore **l'unità di controllo esegue l'istruzione.**

OSSERVAZIONI

Il modello della Macchina di Von Neumann anche se ha ormai quasi cinquanta anni di vita è tuttora adottato dalla maggior parte degli elaboratori.

Il fatto innovativo della Macchina di Von Neumann, che la ha distinta dalle altre macchine di calcolo è che il programma registrato in memoria (stored program computer) insieme ai dati è considerato dall'esecutore a sua volta come se fosse un dato; infatti le istruzioni che lo compongono possono variare durante l'esecuzione del programma adattandosi a risolvere situazioni diverse.

In altre applicazioni c'è ugualmente un programma registrato ma questo è statico e non cambia.

Il suo principale limite è che tutte le operazioni vengono eseguite in stretta sequenza.

Modelli più evoluti prevedono di introdurre varie forme di parallelismo.

LINGUAGGI ASSEMBLATIVI E PROGRAMMAZIONE

Un repertorio di istruzioni assembler plausibile è il seguente:

LOA Carica l’accumulatore A con il contenuto della cella di memoria di indirizzo IND1

ESEMPIO: LOA IND1
MAR←IND1
MDR←read_mem(MAR)
A ← MDR

STO Carica la cella di memoria di indirizzo IND1 con il contenuto del registro accumulatore A

ESEMPIO: STO IND1

MAR ← IND1

MDR ← A

write_mem(MAR,MDR)

IN Trasferimento di dati da una periferica alla memoria centrale

ESEMPIO: IN IND1

MDR ← RDP

MAR ←IND1

write_mem(MAR,MDR)

OUT Trasferimento di dati dalla memoria centrale a una periferica

ESEMPIO: OUT IND1

MAR ← IND1

MDR ← read_mem(MAR)

RDP← MDR

Ove per ipotesi RDP è il registro dati della periferica considerata

ISTRUZIONI ARITMETICHE (ADD, SUB, MUL, DIV)

ADD Somma all’accumulatore A il contenuto della cella di memoria di indirizzo IND1

ESEMPIO: ADD IND1

MAR←IND1

MDR←read_mem(MAR)

OP ← MDR

A ← A+OP

- un operando nel registro OP
- un operando implicitamente nel registro A
- il risultato nel registro A

ISTRUZIONI DI SALTO

Modificano l'esecuzione sequenziale del programma

→ la prossima istruzione da eseguire non è più quella immediatamente successiva, ma quella individuata dall'indirizzo specificato.

Due categorie:

Salto incondizionato

JMP IND1: la prossima istruzione da eseguire è senz'altro quella all'indirizzo IND1.

Salto condizionato

JZ IND1: effettua il salto ad IND1 solo se il contenuto di A è zero

JP IND1: effettua il salto ad IND1 solo se il contenuto di A è positivo

JN IND1: effettua il salto ad IND1 solo se il contenuto di A è negativo

PC ← IND1

ALTRE ISTRUZIONI

HALT termina l'esecuzione del programma

SET DELLE ISTRUZIONI DI UN ELABORATORE

È l'insieme delle istruzioni che la macchina è in grado di eseguire

ESEMPIO 13 istruzioni → 4 bit per l'opcode ($2^4 > 13$)

opcode	istruzione
0000	LOA
0001	STO

0010	IN
0011	OUT
0100	ADD
0101	SUB
0110	MUL
0111	DIV
1000	JMP
1001	JP
1010	JN
1011	JZ
1100	HALT
1101	
1110	
1111	

I simboli IN, OUT, LOA, ecc. si chiamano codici operativi simbolici in quanto servono a ricordare la semantica di ciascuna istruzione.
Come si vede tutte le istruzioni tranne HALT sono ad un solo operando.

Programmazione in Assembler

Un programma in linguaggio macchina consiste di due parti: **istruzioni** e **dati**.
La parte istruzioni precede la parte dati.

ESEMPIO DI PROGRAMMA IN LINGUAGGIO ASSEMBLER

ESEMPIO n°1

Come primo esempio consideriamo un programma che esegue la moltiplicazione di due numeri A e B letti in ingresso e stampa il risultato A*B.

Per semplicità facciamo Ipotesi che:

- le celle di memoria siano da 16 bit
- 4 bit per il codice operativo
- 12 bit per gli operandi
- Indirizzamento: $2^{12} = 4096$ celle di memoria
- il programma parte dalla prima cella di memoria (PC ← 0)
- consideriamo di memorizzare le variabili A,B nelle locazioni di memoria di indirizzo 7 e 8 rispettivamente

0 IN 7

1 IN 8

2 LOA 7
3 MUL 8
4 STO 8
5 OUT8
6 HALT
7 <DATO INTERO A> {16 bit}
8 <DATO INTERO B>

RAPPRESENTAZIONE BINARIA

DEL PROGRAMMA IN LINGUAGGIO ASSEMBLER

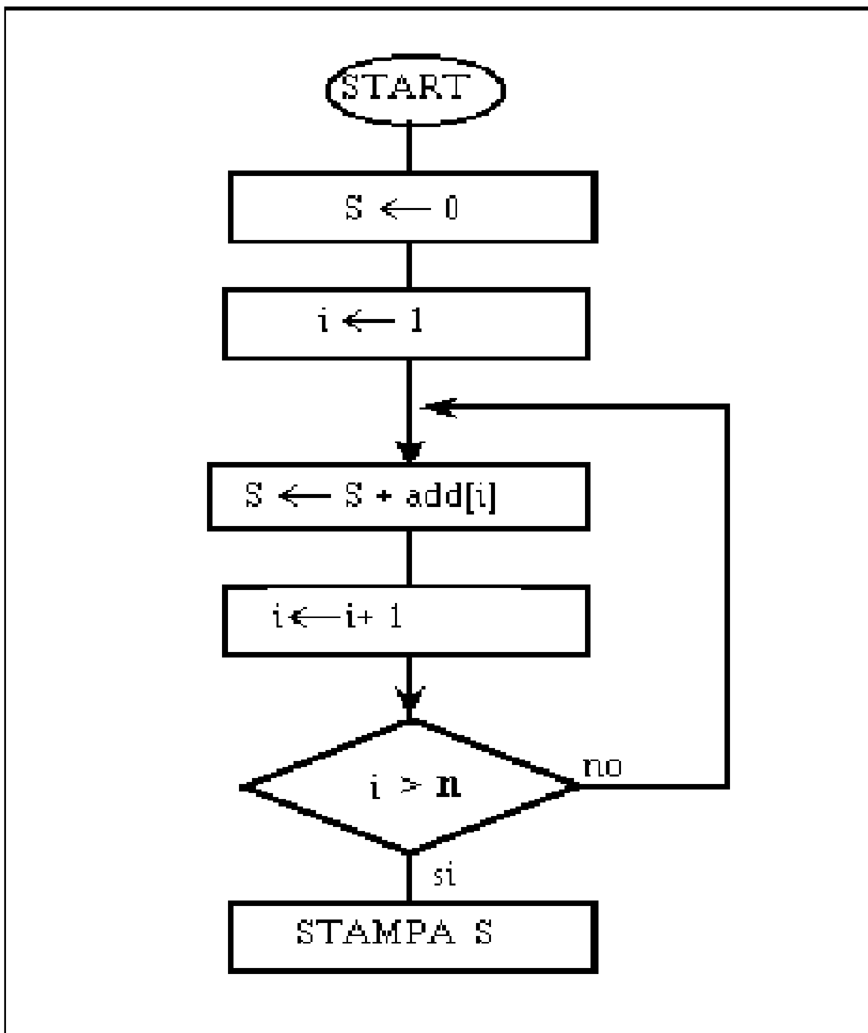
Programma	Rappresentazione binaria
0 IN 7	0 **0010** 000000000111
1 IN 8	1 **0010** 000000001000
2 LOA 7	2 **0000** 000000000111
3 MUL 8	3 **0110** 000000001000
4 STO 8	4 **0001** 000000001000
5 OUT8	5 **0011** 000000001000
6 HALT	6 **0101** 000000001000

7 DATO INTERO A 8 DATO INTERO B	7 **0000** 000000000000 8 **0000** 000000000000

ESEMPIO n°2

Consideriamo un programma che esegue la somma di n numeri e stampa il risultato S.

Un algoritmo che realizza tale calcolo è :



Per semplicità si assumano i valori da sommare già memorizzati nelle locazioni di memoria contigue a partire dall'indirizzo 801, si assuma inoltre il valore di n già memorizzato nella locazione 800

Per codificare l'algoritmo progettato è necessaria una istruzione per l'inizializzazione delle variabili S e i.

Consideriamo allora di modificare l'architettura della macchina e ampliare il set di istruzioni di partenza con la seguente istruzione:

STOV Carica la cella di memoria di indirizzo IND1 con il valore <val>
ESEMPIO: STOV IND1,7
MAR ← IND1
MDR ← 7
write_mem(MAR,MDR)

L'istruzione STOV ha due operandi, uno specifica in modo ***diretto*** l'indirizzo di una locazione di memoria, l'altro specifica in modo ***immediato*** il valore dell'operando.
Il primo tipo di riferimento all'operando viene detto *indirizzamento diretto*, il secondo *indirizzamento immediato*.

Tale istruzione, avendo due operandi, va gestita in modo adeguato, in quanto probabilmente necessita di due locazioni di memoria contigue per essere registrata, una per il codice operativo STOV ed il primo operando e l'altra per il secondo operando, pertanto al termine della esecuzione di tale istruzione il PC deve essere incrementato di 2.

Supponendo che si usi la locazione di indirizzo 790 serva per la memorizzazione della somma S, la locazione di indirizzo 791 per la memorizzazione del valore costante 1, il programma che implementa tale algoritmo, memorizzato a partire dalla locazione 378, è il seguente:

378 STOV 790,0 inizializza S a 0
380 STOV 792,1 inizializza i a 1

382 STOV 791,1 memorizza la costante 1

****

384 LOA 790 carica S nell'accumulatore

385 ADD 801 incrementa l'accumulatore con il contenuto della locazione di indirizzo 801

386 STO 790 aggiorna S

****

387 LOA **385** carica l'istruzione di indirizzo 385 nell'accumulatore

388 ADD 791 incrementa l'accumulatore di uno (per considerare l'addendo successivo,

ovvero quello della locazione 802, 803..) i<-i+1

389 STO 385 aggiorna l'istruzione di indirizzo 385

****

390 LOA 800 carica n nell'accumulatore

391 MIN 791 decrementa l'accumulatore di 1

392 STO 800 aggiorna n

****

393 JP 384 se n>0 torna all'istruzione 384

****

394 OUT 790 stampa S

395 STOP

....

790 <---S

791 <---1

792 <---i

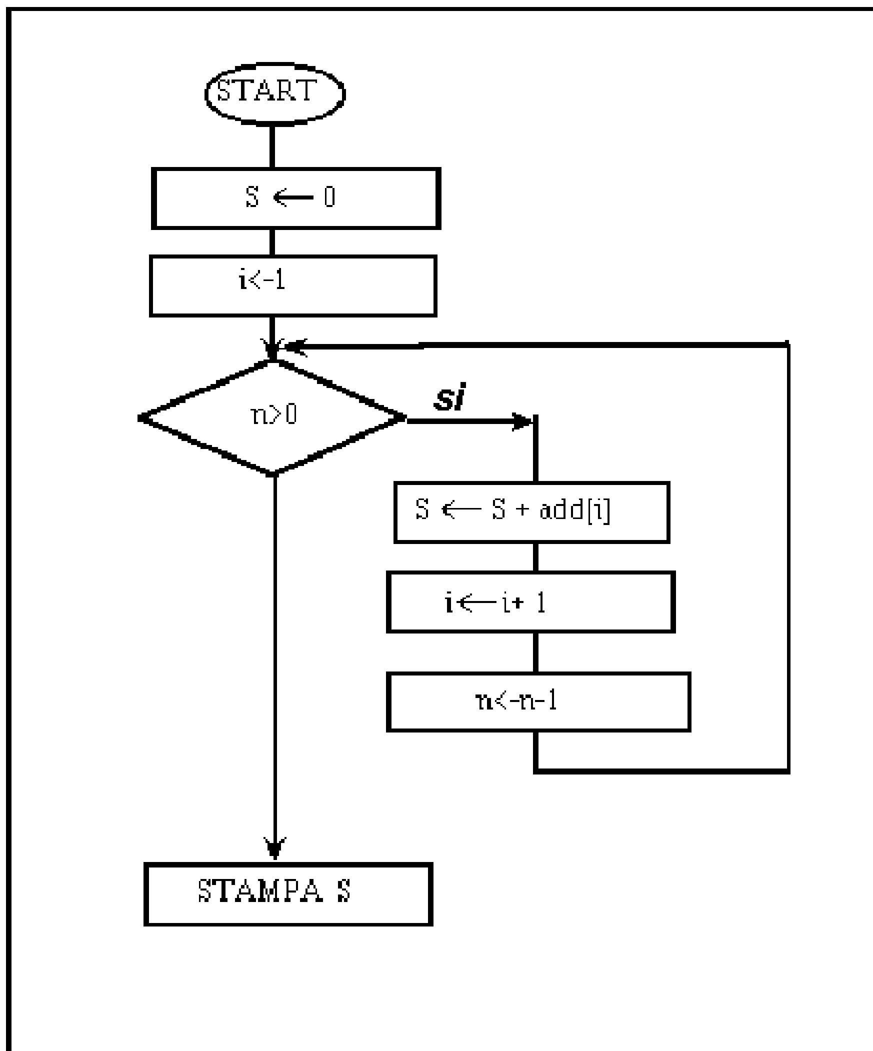
800 <-n

801 <-A1

....

800+n <-An

Un altro algoritmo per eseguire lo stesso calcolo è:



ad esso corrisponde il programma, di seguito riportato, che si suppone memorizzato a partire dalla locazione 378.

378 STOV 790,0 inizializza S a 0

380 STOV 791,1 memorizza la costante 1

382 LOA 800 carica n nell'accumulatore

383 JZ 394 se l'accumulatore vale 0 vai a fine

****

384 LOA 790 carica S nell'accumulatore

385 ADD 801 incrementa l'accumulatore con il contenuto della locazione di indirizzo 801

386 STO 790 aggiorna S

****

387 LOA **385** carica l'istruzione di indirizzo 385 nell'accumulatore

388 ADD 791 incrementa l'accumulatore di uno (per considerare l'addendo successivo, ovvero quello della locazione 802,803...)

389 STO 385 aggiorna l'istruzione di indirizzo 385

****

390 LOA 800 carica n nell'accumulatore

391 MIN 791 decrementa l'accumulatore di 1

392 STO 800 aggiorna n

****

393 JMP 383 torna all'istruzione 383

394 OUT 790 stampa S

395 STOP

....

790 <---S

791 <---1

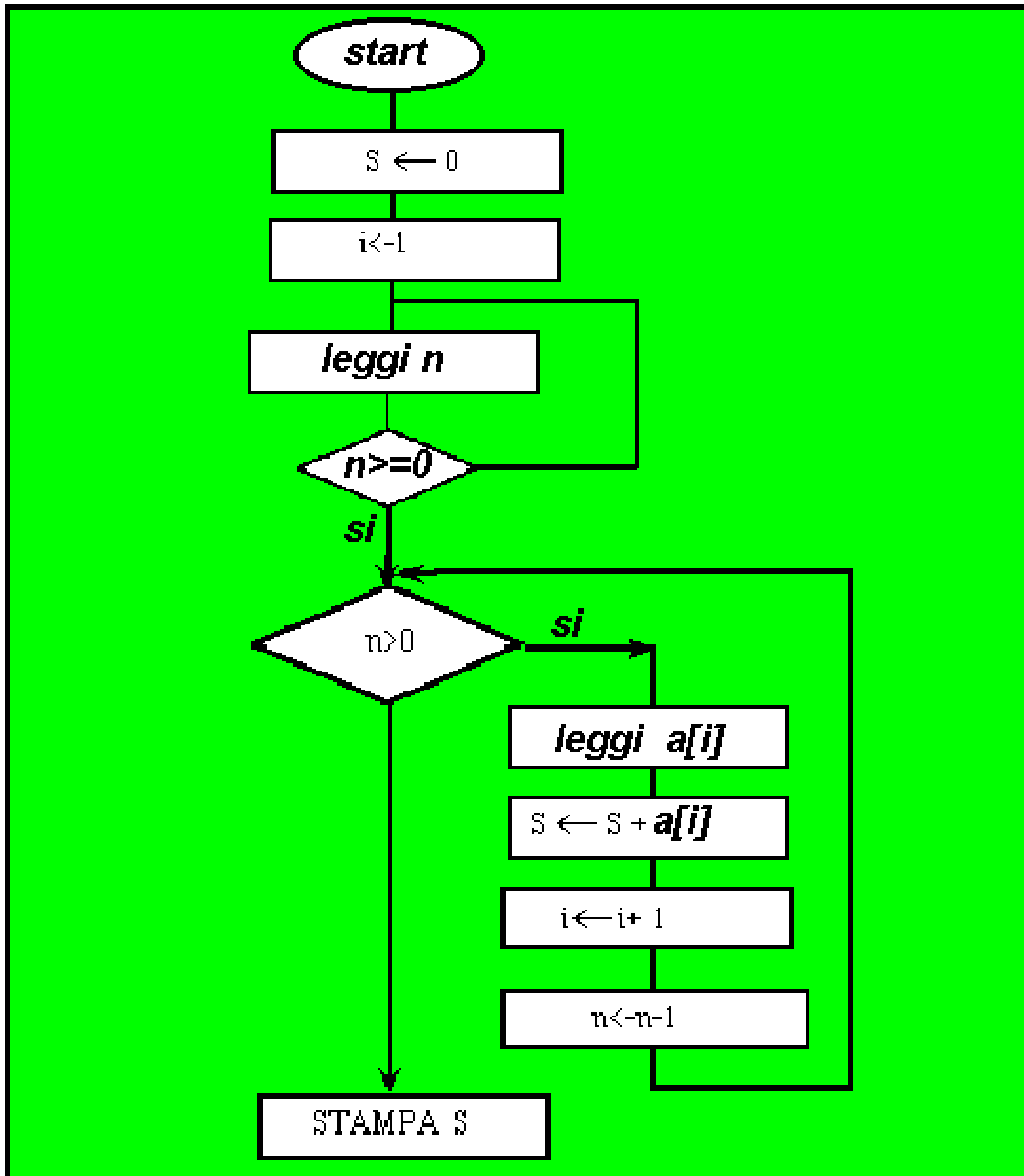
800 <-n

801 <-A1

....

800+n <-An

Il programma completo con la lettura e memorizzazione dei dati da sommare è:



372 STOV 790,0 inizializza S a 0

374 STOV 791,1 memorizza la costante 1

376 IN 800 leggi da input n

377 LOA 800 carica n nell'accumulatore

378 JN 376 se n<0 torna indietro

379 JZ 394 se l'accumulatore vale 0 vai a fine

****

380 IN 801 Leggi a[i]

381 LOA 801

382 ADD 790 incrementa l'accumulatore con il contenuto della locazione di indirizzo 790

(S)

383 STO 790 aggiorna S

****

384 LOA 380 mi preparo a memorizzare il dato nella locazione successiva

385 ADD 791

386 STO 380

****

387 LOA 381 mi preparo a sommare il dato dalla locazione successiva

388 ADD 791

389 STO 381

****

390 LOA 800 carica n nell'accumulatore

391 SUB 791 decrementa l'accumulatore di 1

392 STO 800 aggiorna n

****

393 JMP 379 torna all'istruzione 379

394 OUT 790 stampa S

395 HALT

....

790 <---S

791 <---1

800 <-n

801 <-A1

....

800+n <-An

....

790 <---S

791 <---1

800 <-n

801 <-A1

....

800+n <-An

OSSERVAZIONI

Questi esempi mostrano la potenza della macchina di Von Neumann, dove istruzioni e dati di un programma vengono trattati alla stessa stregua.

Infatti nelle istruzioni LOA e STO ad un operando, il campo operando corrisponde ad un indirizzo di memoria che può essere riferito sia a un operando vero e proprio sia a una istruzione.

Fino ad ora abbiamo visto due modi di indirizzamento degli operandi nelle istruzioni: l'*indirizzamento immediato* , che si ha quando il campo operando di una istruzione contiene l'operando stesso e l'*indirizzamento diretto* , che si ha quando il campo operando di una istruzione contiene l'indirizzo della locazione di memoria che contiene l'operando. Negli esempi successivi vedremo altre tecniche di indirizzamento.
Indirizzamento indiretto quando il campo operando contiene l'indirizzo della locazione di memoria che a sua volta contiene l'indirizzo della locazione di memoria dove risiede l'operando.
Indirizzamento tramite registro indice quando l'indirizzo dell'operando si ottiene sommando un valore numerico ad un registro della CPU detto registro indice.

INDIRIZZAMENTO

L'operando di una istruzione può rappresentare:

· il dato stesso su cui operare

→ INDIRIZZAMENTO IMMEDIATO

· l'indirizzo della cella di memoria in cui si trova il dato

· direttamente → INDIRIZZAMENTO DIRETTO

· indirettamente → INDIRIZZAMENTO INDIRETTO

· tramite un registro ausiliario ("registro indice")

→ INDIRIZZAMENTO INDICIZZATO

€

INDIRIZZAMENTO IMMEDIATO

L’operando **rappresenta già il valore da usare** nell’operazione.

Non è richiesto alcun accesso alla memoria durante l'esecuzione dell'istruzione.

Pro:

· semplice e veloce

Contro:

· con istruzioni a formato fisso, ***gli operandi possono occupare al massimo m bit***

· l’operando deve essere ***noto a priori***

· l’operando risulta ***interno al programma e non parametrico (cablato)***

INDIRIZZAMENTO DIRETTO

Il campo operando contiene **l’indirizzo assoluto** della cella di memoria che contiene il dato.
È richiesto un accesso alla memoria durante l’esecuzione dell’istruzione, per recuperare il dato.

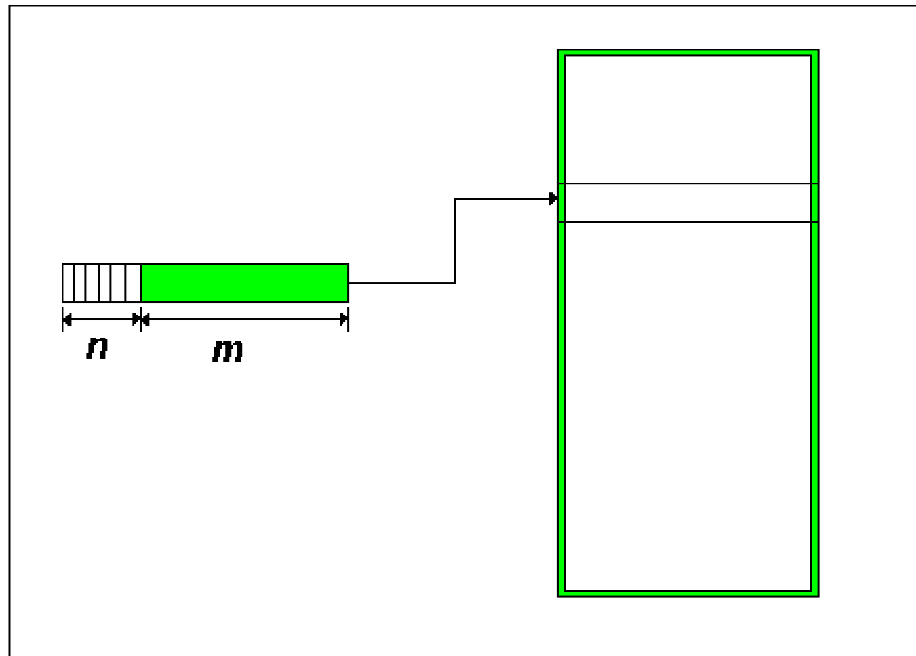
Pro:

· il dato è parametrico

· il dato non deve avere dimensione fissa

· il dato non deve per forza essere noto a priori

Contro:



· può essere oneroso se la memoria è molto grande o se è necessario rilocare il programma ed i dati.

ESEMPIO n°3

Per risolvere in modo più semplice lo stesso problema dell'esempio n°2 si può immaginare di modificare l'architettura della macchina ampliando il set di istruzioni di partenza con la seguente istruzione:

ADD* Somma all'accumulatore A il contenuto della cella di memoria il cui indirizzo è scritto nella locazione di indirizzo IND1

ESEMPIO: ADD* IND1

MAR←IND1

MDR←read_mem(MAR)

MAR ← MDR

MDR←read_mem(MAR)

OP ← MDR

A ← A+OP

Tale istruzione permette il cosiddetto *indirizzamento indiretto* degli operandi. Secondo tale tecnica il campo operando della istruzione (contrassegnata da *) contiene l'indirizzo della locazione di memoria che a sua volta contiene l'indirizzo della locazione di memoria dove risiede l'operando.

INDIRIZZAMENTO INDIRETTO

L'operando contiene **non più** l'indirizzo della cella di memoria che contiene il dato, ma **l'indirizzo assoluto di una cella di memoria che a sua volta contiene l'indirizzo** a cui si trova il dato.

È richiesto un doppio accesso alla memoria durante l'esecuzione dell'istruzione, per recuperare il dato.

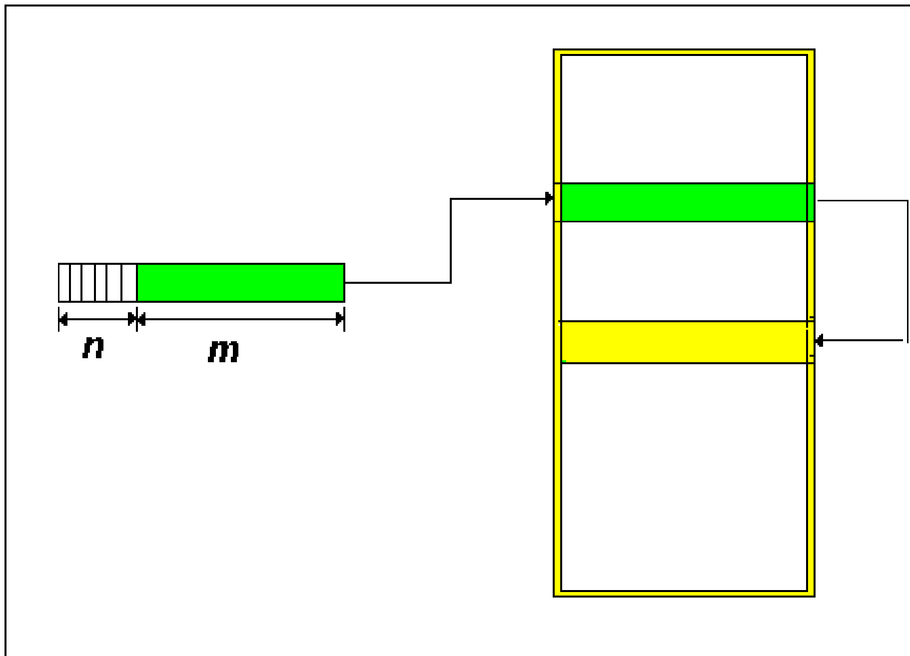
Pro:

· si possono usare più bit per rappresentare l'indirizzo dell'operando → lo spazio indirizzabile aumenta

· è semplice rilocare i dati altrove

Contro:

· intrinsecamente più lento causa il doppio accesso alla memoria.



Ferme restando le locazioni di memoria impegnate per la rappresentazione dei dati e supponendo di dedicare la locazione di memoria di indirizzo 792 alla memorizzazione dell'indirizzo dell'addendo da sommare, il programma che esegue tale calcolo potrebbe essere il seguente:

376 STOV 790,0 inizializza S a 0

378 STOV 791,1 inizializza i a 1

380 **STOV 792,801** inizializza la locazione 792 con 801 (indirizzo della locazione del primo addendo)

382 LOA 800 carica n nell'accumulatore

383 JZ 394 se l'accumulatore vale 0 vai a fine

****

384 LOA 790 carica S nell'accumulatore

385 ADD* 792 incrementa S con il contenuto della locazione
il cui indirizzo si trova nella locazione di indirizzo 792

386 STO 790 aggiorna S

****

387 LOA 792 carica la locazione di indirizzo 792 nell'accumulatore (ove si trova l’indirizzo di a[i])

388 ADD 791 incrementa l'accumulatore di uno (per considerare l'addendo successivo)

389 STO 792 aggiorna la locazione di indirizzo 792

****

390 LOA 800 carica n nell'accumulatore

391 MIN 791 decrementa l'accumulatore di 1

392 STO 800 aggiorna n

****

393 JMP 383 torna all'istruzione 383

394 OUT 790 stampa S

395 STOP

....

790 <---S

791 <---1

792 <---801

800 <-n

801 <-A1

....

800+n <-An

ESEMPIO n°4

La soluzione presentata nell'esempio precedente ha come *aspetto negativo* quello di *richiedere accessi doppi in memoria*, infatti prima di accedere all'operando vero e proprio è necessario accedere in memoria per calcolare l'indirizzo dell'operando.
Per ovviare a tale inconveniente si può immaginare nuovamente di modificare l'architettura della macchina, ampliando il set delle istruzioni di partenza con le istruzioni che seguono, prevedendo altri registri, detti *registri indice* con le stesse funzioni dell’accumulatore ma che permettono anche un’altra modalità di indirizzamento più efficiente.

- Istruzione che permette l'inizializzazione del registro indice RX

LOAX Carica il registro indice RX con il contenuto della cella di memoria di indirizzo IND1

ESEMPIO: LOAX IND1,RX

MAR←IND1

MDR←read_mem(MAR)

RX ← MDR

- Istruzione che consente di utilizzare i registri indice come accumulatori.

ADDX Somma al registro indice RX (considerato come accumulatore) il contenuto della cella di memoria di indirizzo IND1.

ESEMPIO: ADDX IND1,RX

MAR←IND1

MDR←read_mem(MAR)

OP ← MDR

RX ← RX+OP

- un operando nel registro OP
- un operando implicitamente nel registro RX
- il risultato nel registro RX

- Istruzione che permette l'indirizzamento degli operandi tramite il registro indice RX

ADD$ Somma all'accumulatore A il contenuto della cella di memoria di indirizzo pari al contenuto della locazione di memoria di indirizzo IND1 aumentato del contenuto del registro indice RX

ESEMPIO: ADD$ IND1,RX

MAR←IND1+RX

MDR←read_mem(MAR)

OP ← MDR

A ← A+OP

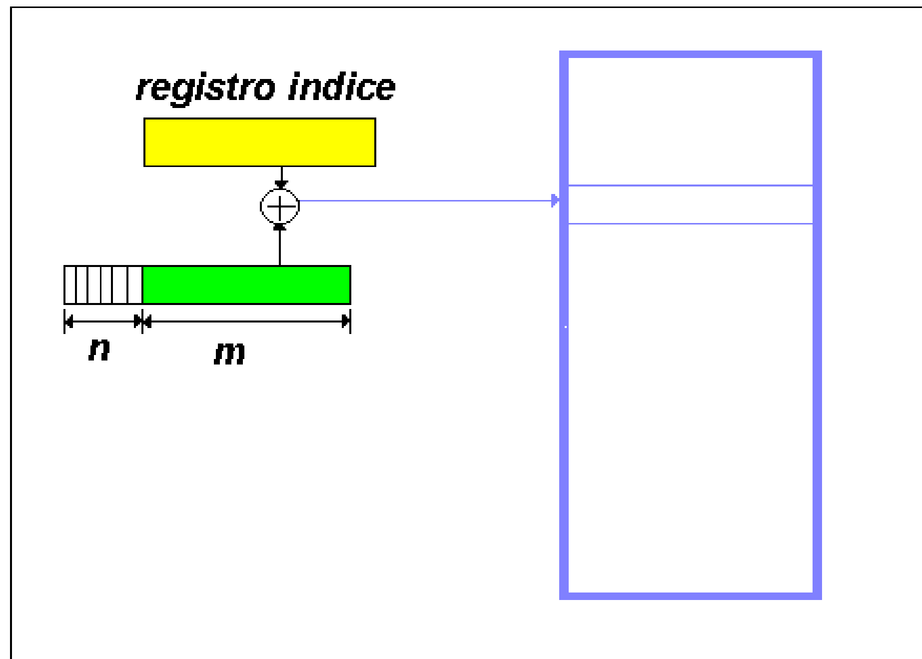
INDIRIZZAMENTO INDICIZZATO

L’indirizzo dell’operando si ottiene **sommando** algebricamente l’operando dell’istruzione col contenuto di un particolare registro della CPU, detto **registro indice.**

Richiede un unico accesso alla memoria per recuperare l’operando, più un’operazione di somma.

Pro:

· molto efficiente accedere a dati memorizzati in celle di memoria consecutive (vettori, stringhe, ..)



· più semplice gestire la rilocazione

Un programma per risolvere lo stesso problema, ferme restando le locazioni di memoria impegnate per la rappresentazione dei dati, basato sull'uso di un registro indice (RX) e delle nuove istruzioni, potrebbe essere il seguente:

379 STOV 790,0 inizializza S a 0

380 STOV 791,1 inizializza ctr a 1

381 LOAX 790,RX carica 0 nel registro indice RX (RX <-Mem[790])

(RX serve per indirizzare l'addendo da sommare a S; ind.add=

801+RX)

382 LOA 800 carica n nell'accumulatore

383 JZ 393 se l'accumulatore vale 0 vai a fine

****

384 LOA 790 carica S nell'accumulatore

385 **ADD$ 801,RX** incrementa l'accumulatore con il contenuto della locazione

il cui indirizzo è pari a 801
incrementato del registro indice 3

(A← A+Mem[801+RX])

Uso il registro indice RX per

l'indirizzamento

386 STO 790 aggiorna S

****

388 **ADDX 791,RX** incrementa il registro indice di uno (per considerare l'addendo successivo)

(RX← RX +Mem[791])

Uso RX come accumulatore

****

389 LOA 800 carica n nell'accumulatore

390 MIN 791 decrementa l'accumulatore di 1

391 STO 800 aggiorna n

****

392 JMP 383 torna all'istruzione 383

393 OUT 790 stampa S

394 STOP

....

790 <---S

791 <---1

800 <-n

801 <-A1

....

800+n <-An

Negli esempi riportati abbiamo visto quattro tecniche fondamentali di indirizzamento degli operandi: l'*indirizzamento immediato* (quando il campo operando di una istruzione contiene l'operando stesso), l'*indirizzamento diretto* (quando il campo operando di una istruzione contiene l'indirizzo della locazione di memoria che contiene l'operando), l'*indirizzamento indiretto* (quando il campo operando contiene l'indirizzo della locazione di memoria che a sua volta contiene l'indirizzo della locazione di memoria dove risiede l'operando) e l'*indirizzamento tramite registro indice* (quando l'indirizzo dell'operando si ottiene sommando un valore numerico ad un registro della CPU detto registro indice). Quest'ultimo tipo di indirizzamento si presta bene a scorrere in modo efficiente una tabella di dati, posti in memoria in locazioni contigue, infatti nel registro indice si memorizza lo scostamento dell'elemento da sommare rispetto al primo elemento e con successivi incrementi del registro indice si genera l'indirizzo degli operandi successivi.

SET DI ISTRUZIONI DI UN ELABORATORE

L'insieme delle istruzioni che la macchina è in grado di eseguire può essere molto ridotto (RISC) o estremamente ampio (CISC).

I linguaggi macchina dei microprocessori attuali (come il Pentium) sono molto più ricchi (istruzioni più numerose e complesse, più registri, ecc.)
→ macchine **CISC** **(Complex Instruction Set Computer)**

Ci sono invece, come SPARC e PowerPC, caratterizzate da un ridotto set di istruzioni con formati regolari
→ macchine **RISC** **(Reduced Instruction Set Computer)**

Le prestazioni dei RISC sono spesso migliori rispetto a quelle delle macchine CISC.

IL LINGUAGGIO MACCHINA

Leggere e capire un programma scritto in forma binaria è difficile:

0 **0010** 000000000111
1 **0010** 000000001000
2 **0000** 000000000111

3 **0110** 000000001000
4 **0001** 000000001000
5 **0011** 000000001000
6 **0101** 000000001000
7 **1100** 000000000000
8 0000 000000000000

LINGUAGGI ASSEMBLATORI (ASSEMBLER)

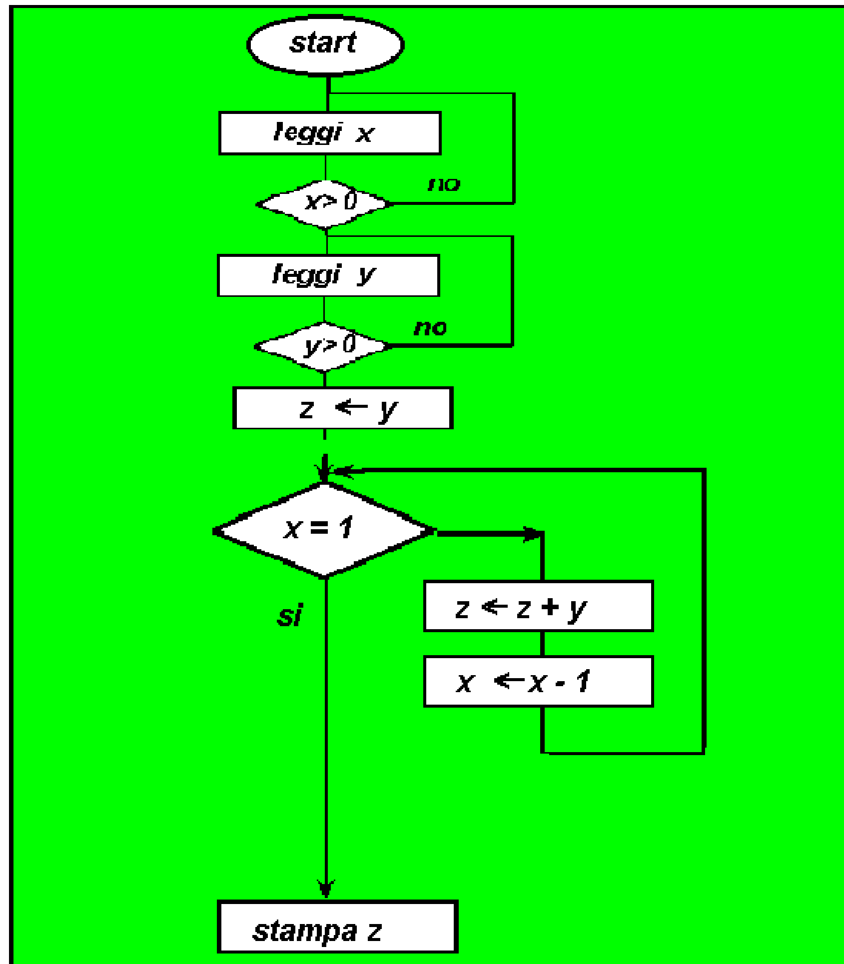
Linguaggi le cui istruzioni corrispondono **univocamente** a quelle del linguaggio macchina, ma sono espresse tramite **nomi simbolici** (parole chiave) invece che in binario.

- I riferimenti alle celle di memoria sono fatti mediante nomi simbolici (identificatori).
- Identificatori che rappresentano **dati** (costanti o variabili) oppure istruzioni (**etichette**).

Il programma prima di essere eseguito deve essere tradotto in linguaggio macchina
→ **Assemblatore**

ESEMPIO

Programma che calcola il prodotto tra due numeri per mezzo di somme successive




```
LX      IN X
          LOA X
          JN LX
LY      IN Y
          LOA Y
          JN LY
          LOA Y
          STO Z
TEST    LOA X
          SUB UNO
          STO X
JZ      FINE
          LOA Z
          SUM Y
          STO Z
          JMP TEST
FINE    OUT Z
          HALT
```

ZERO 0

UNO 1

X INT

Y INT

Z INT

COME ESEGUIRE UN PROGRAMMA SCRITTO IN LINGUAGGIO SIMBOLICO?

Il modo di eseguire i programmi scritti nei linguaggi simbolici (ad alto livello) segue principalmente due vie, quella degli interpreti e quella dei compilatori.

L'**interprete** di un linguaggio è un programma che,

- legge una alla volta le istruzioni di un programma sorgente,
- verifica la correttezza sintattica della istruzione, sulla base della sintassi del linguaggio,
- in caso positivo (assenza di errori), sulla base della semantica del linguaggio, la traduce nella corrispondente sequenza di istruzioni in linguaggio macchina e
- la esegue.

Il **compilatore** di un linguaggio è un programma che,

- dopo aver verificato la correttezza sintattica del programma sorgente, sulla base della sintassi del linguaggio,
- traduce il programma sorgente nel programma oggetto (sulla base della semantica del linguaggio), il quale potrà successivamente essere eseguito.

Pertanto quando si vuole eseguire un programma scritto con un linguaggio interpretato, l'interprete deve essere caricato nella macchina per provvedere ad interpretare e ad eseguire sotto forma di istruzioni della CPU ciò che è stato scritto, mentre quando si vuole eseguire un programma scritto con un linguaggio compilato, l'esecuzione del programma deve essere preceduta da una sua traduzione in linguaggio macchina.
(Il compilatore di un linguaggio assembly viene detto assemblatore).

VERSO LINGUAGGI DI PROGRAMMAZIONE SIMBOLICI AD ALTO LIVELLO

Linguaggio Macchina

• Conoscenza precisa dei metodi di rappresentazione e manipolazione delle informazioni utilizzati

Linguaggio Macchina ed Assembler

• Necessità di conoscere dettagliatamente le caratteristiche della macchina (registri, dimensioni dati, set di istruzioni)

• Semplici algoritmi richiedono l'uso di molte istruzioni

Linguaggi di Alto Livello

Il programmatore può astrarre dai dettagli legati all'architettura e può esprimere i propri algoritmi in modo simbolico.

I linguaggi di alto livello sono **indipendenti dalla macchina fisica**.

COME ESEGUIRE UN PROGRAMMA SCRITTO IN UN LINGUAGGIO DI ALTO LIVELLO?

Occorre **tradurlo** nel linguaggio macchina dello specifico processore che si sta usando.

Due modalità possibili:

• **compilazione** (es. C,C++, FORTRAN, Pascal, ..)

• **interpretazione** (es. Basic, Perl, Java, …)

I **compilatori** traducono un intero programma dal linguaggio L al linguaggio macchina della macchina prescelta

- traduzione e esecuzione procedono separatamente
- al termine della compilazione è disponibile la versione tradotta del programma
- la versione tradotta è però specifica di quella macchina
- per eseguire il programma basta avere disponibile la versione tradotta (non serve il programma sorgente!)

Gli **interpreti** invece traducono e immediatamente eseguono il programma **istruzione per istruzione**

- traduzione ed esecuzione procedono insieme
- al termine non vi è alcuna versione tradotta del programma originale
- se si vuole rieseguire il programma occorre anche ritradurlo.

FASI DI SVILUPPO DI UN PROGRAMMA

Qualunque sia il linguaggio di programmazione scelto occorre:

• Scrivere il **testo del programma** e memorizzarlo su supporti di memoria permanenti (editing);

Se il linguaggio è compilato:

- Tradurre il linguaggio in **linguaggio macchina** (compilazione);
- Eseguire il programma tradotto.

Se il linguaggio è interpretato:

- Usare l'interprete per eseguire il programma.