U.D. 1

2 La struttura dell’elaboratore

Fonti: Antola Dispense del Corso

Esecuzione dei programmi

Esecuzione «manuale» di un programma

Istruzioni da eseguire

```
.....
int w, y, z;
int sp, ns;

leggi w;
leggi y;

sp=0;
ns=y;

while (ns >0)
 {
  sp=sp+w;
  ns=ns-1;
 }
z=sp;
scrivi z;
```

Nella tabella è riportata la traccia di esecuzione con i valori assunti dalle variabili nei punti significativi *A, B, .. F*
(si supponga che i dati in ingresso siano 7 e 3)

Punti della traccia di esecuzione

	A	*B*	*C*	*D*	*E*	*F*
w	??	7	7	7	7	7
y	??	3	3	3	3	3
sp	??	0	7	14	21	21
ns	??	3	2	1	0	0
z	??	??	??	??	??	21

A: valore dopo la dichiarazione
B: valore dopo l'inizializzazione
C: valore dopo la 1ma iterazione del ciclo
D: valore dopo la 2nda iterazione del ciclo
E: valore dopo l'ultima iterazione del ciclo
F: valore al termine del programma

Istruzioni: «in memoria» scritte e leggibili
Variabili: «in memoria» scrivibili e leggibili

Esecutore calcolatore: requisiti funzionali

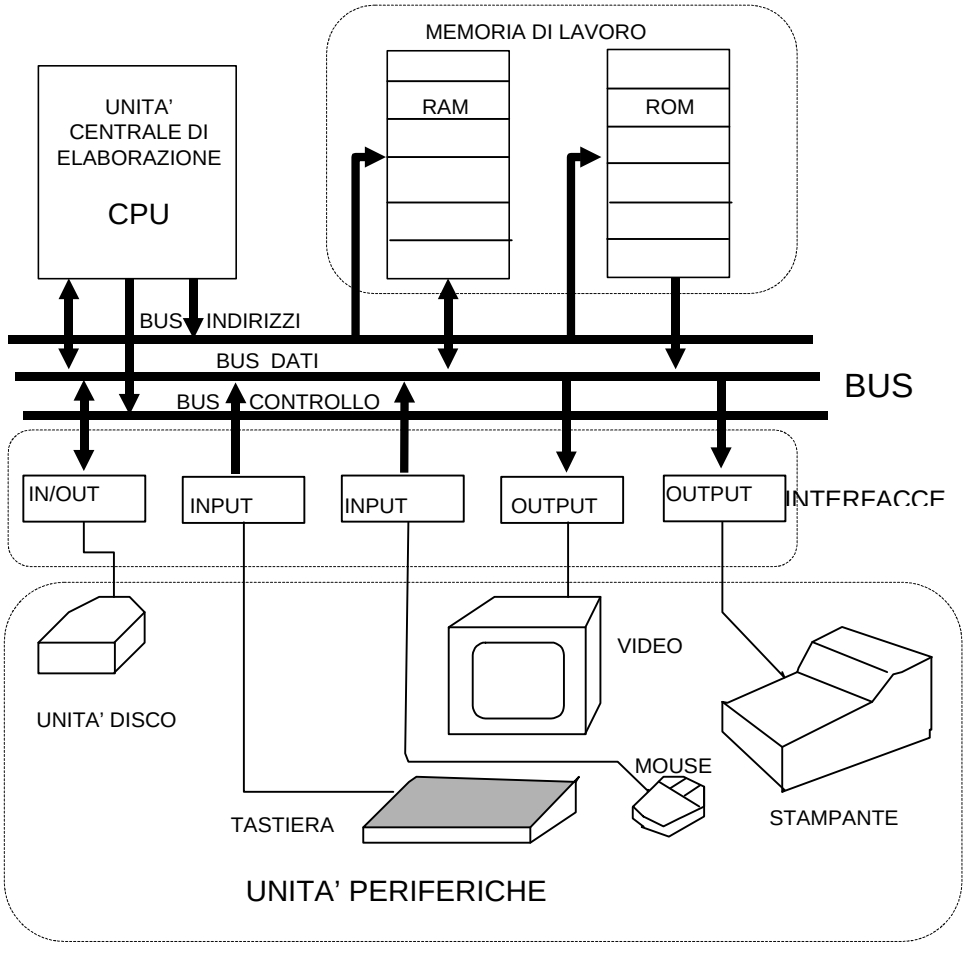
- capire ed eseguire le istruzioni
  ⇒ **unità centrale** di elaborazione
  (CPU - Central Processing Unit)
- mantenere istruzioni e dati
  ⇒ **memoria di lavoro**
- interagire con «mondo esterno»
  ⇒ **interfacce di I/O**

Architettura di Eckert - Mauchly - Von Neumann

Può' essere considerato un **modello tipico** dell'architettura interna di un calcolatore.

E' costituito da **3 blocchi funzionali** collegati tra loro tramite un **BUS** (insieme di collegamenti elettrici) che consente il trasferimento di informazioni tra i blocchi.

Schema dell'architettura tipica di Calcolatore



UNITÀ CENTRALE DI ELABORAZIONE (*Central Processing Unit* - CPU):

- per eseguire un programma è necessario che questo sia caricato (in formato eseguibile) nella **memoria di lavoro**.

- la CPU **legge da memoria, interpreta ed esegue le istruzioni** del programma operando sulle variabili (dati). Può essere considerata l'unità attiva del calcolatore.

- la CPU per leggere ed eseguire le istruzioni **gestisce, controlla e temporizza** il funzionamento delle altre unità (memorie e interfacce) tramite i segnali del bus.

- le **istruzioni** devono essere espresse in **codice macchina**, cioè nel formato direttamente interpretabile dalla CPU.
  Ogni istruzione è quindi rappresentata in codice macchina da una sequenza di 0 e 1 che la identifica in modo univoco.

- le **variabili** devono essere «espresse» in modo da essere accessibili dalla CPU:
  il **riferimento** ad una variabile è rappresentato da un **indirizzo** della memoria di lavoro.
  Il **valore** della variabile è contenuto nella parola di memoria associata all'indirizzo ed è rappresentato tramite una **codifica binaria** opportuna, dipendente dal tipo di variabile.

Memoria di lavoro - 1

- è un insieme ordinato di parole (celle) che possono contenere (memorizzare) informazioni, e cioè **istruzioni** e **dati**

  La memoria di lavoro può essere pensata come una tabella monodimensionale, nella quale gli elementi sono le **parole** di memoria

- una parola di memoria è costituita da h **elementi di memoria binari** (ad es. h = 8, **16**, **32**, 64 bit).

Esempio di parola da 16 bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	1	1	0	0	0	1	1	0	1

- la posizione di ogni parola di memoria è identificata in modo univoco da un numero intero positivo, detto **indirizzo** della parola di memoria

- per **accedere** ad una generica parola di memoria è necessario fornire il suo indirizzo (che la identifica in modo univoco)

- se k è il numero di bit utilizzabili per specificare l'indirizzo, allora 2^k è l'area di memoria fisica indirizzabile. Ad esempio, con 23 bit di indirizzo lo spazio di indirizzamento è di 8Mega parole.

Memoria di lavoro - 2

- la memoria di lavoro è detta ad *accesso casuale*, perché data una parola cui si accede (e quindi il suo indirizzo) la prossima parola cui si può accedere può essere in posizione «qualsiasi». In contrapposizione con *accesso sequenziale* .....

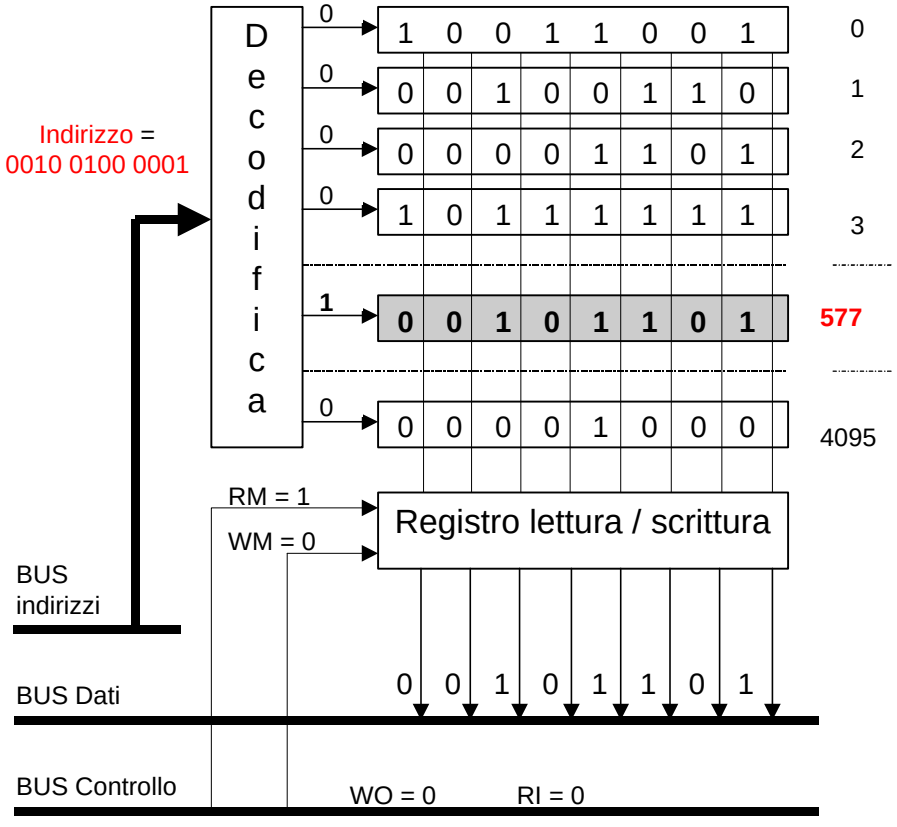
- la memoria di lavoro è, in generale, composta da due tipi di memorie: la memoria **RAM** (*Random Access Memory*) la memoria **ROM** (*Read Only Memory*).

- le parole di memoria RAM sono modificabili, cioè leggibili e scrivibili. La memoria ROM è solo leggibile: le informazioni sono scritte in modo «permanente» di solito dal costruttore, prima dell'installazione nel calcolatore.

Caratteristiche tecnologiche:

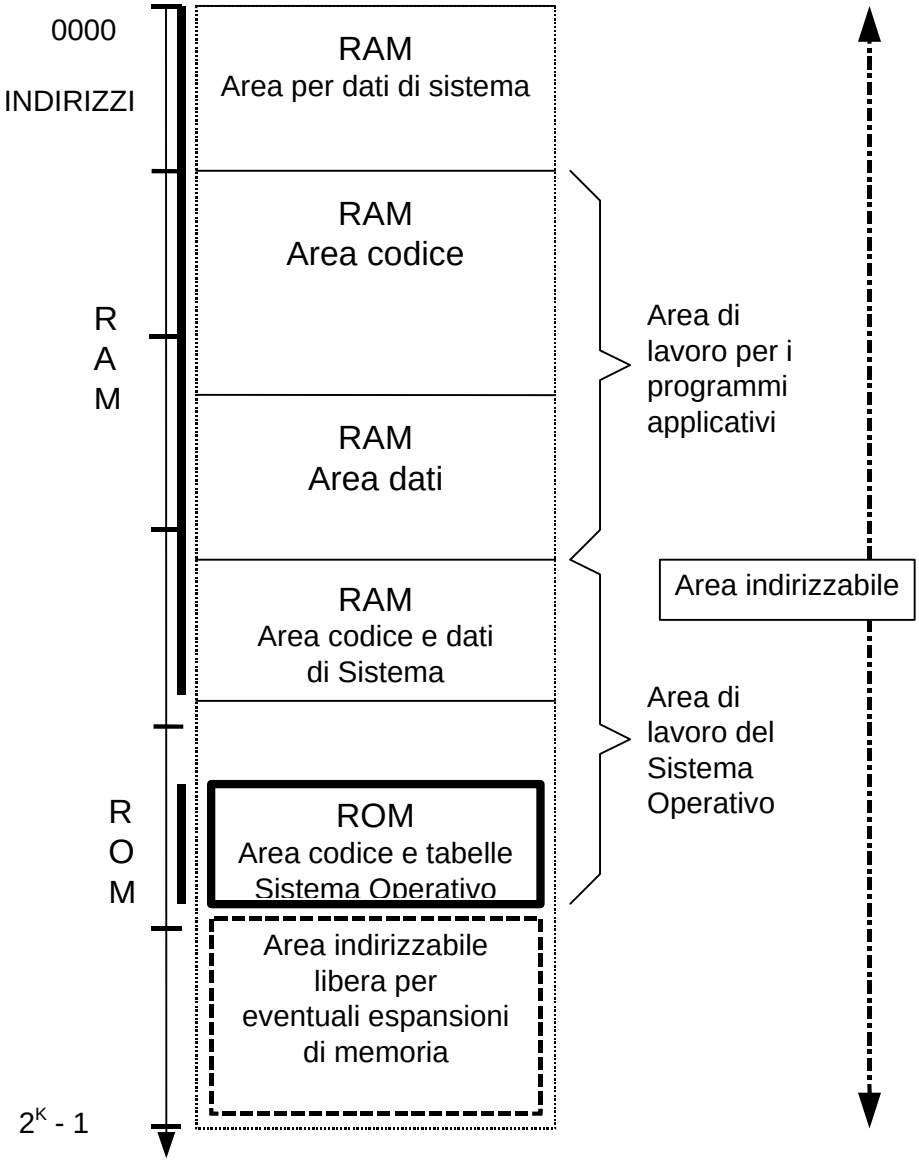
⇒ è realizzata con circuiti a transistori
⇒ la memoria RAM è *volatile* e quindi mantiene le informazioni solo se alimentata.
⇒ all'accensione il contenuto delle parole di memoria RAM è una sequenza casuale di «zeri» e «uni»
⇒ nella memoria ROM le informazioni sono permanentemente scritte e non modificabili
⇒ i tempi di accesso alla singola parola di memoria sono dell'ordine delle decine di nanosecondi
⇒ è una memoria ad accesso veloce rispetto alla memoria di massa (tempi di accesso dell'ordine delle decine di millisecondi e quindi circa 1 milione di volte più lenta) ma ha un numero di parole inferiore (dell'ordine dei Megabyte)

Schema funzionale della memoria di lavoro (RAM)



BUS Indirizzi di 12 bit (da 0 a 4095)
BUS Dati e parole di memoria di 8 bit
Nello schema è evidenziato il caso di **lettura da memoria**
all'indirizzo 577 che in binario su 12 bit è 0010 0100 0001 HEX 0241h
il cui contenuto letto è 0010 1101 HEX 02Dh

Partizione della memoria



Interfacce di ingresso/uscita (*Input/Output* - I/O) - 1

Sono dispositivi circuitali che consentono al calcolatore di scambiare informazioni con le unità periferiche (dispositivi elettromeccanici).

Tipiche unità periferiche sono la tastiera e il mouse (di ingresso), e il video e la stampante (di uscita).

E' definita come unità periferica anche la **memoria di massa** (ad esempio l'unità a disco fisso, l'unità a floppy) che è di ingresso e di uscita e non comunica con l'esterno.

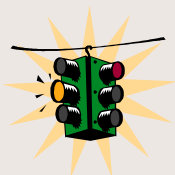
Struttura delle interfacce di ingresso/uscita

Le interfacce di I/O sono costituite da *porte di I/O* e da circuiti ausiliari. Le porte sono **registri** (simili a parole di memoria) leggibili e/o scrivibili da parte della CPU, ma comunicanti anche con il mondo esterno (periferiche) con segnali in ingresso e in uscita.

Considerando uno schema molto semplificato, un'interfaccia è costituita dai seguenti registri.

- Registro **dati** periferica
- Registro **comandi** periferica
- Registro di **stato** periferica

Azioni di I/O

- Due mondi con velocità diverse
  - Periferiche meccaniche – lente
  - CPU veloce
- I due mondi si devono sincronizzare
- Produttore – Consumatore
  - Prod lento (Input) → Cons attende 
  - Prod veloce (Output) → Cons blocca 

Interfacce di ingresso/uscita - 2

Le funzionalitàdei registri di un'interfaccia sono le seguenti:

Registro Dati Periferica (RDP):

- nel caso di i**nterfaccia di ingresso** il registro rappresenta il «luogo fisico» in cui l'unità periferica (elettromeccanica) fornisce i dati perché questi possano essere acquisiti (cioè letti) dalla CPU

- nel caso di i**nterfaccia di uscita**: il registro rappresenta il «luogo fisico» da cui l'unità periferica (elettromeccanica) riceve i dati che sono stati generati (cioè scritti) dalla CPU

Registro Comandi Periferica (RCP)

il registro di comando contiene le informazioni (in codifica binaria) che determinano il *modo di funzionamento* della periferica. Tali informazioni sono generate (scritte) dalla CPU e utilizzate dalla periferica.

Registro di Stato Periferica

il registro di stato contiene le informazioni (in codifica binaria) relative allo *stato di funzionamento* della periferica. Tali informazioni sono generate (scritte) dalla periferica e utilizzate (lette) dalla CPU.

Bus di sistema - 1

E' il sistema di comunicazione che consente il **trasferimento** delle informazioni dalla memoria di lavoro e dalle interfacce di I/O alla CPU e viceversa.

E' costituito da un numero adeguato di conduttori e ogni conduttore «trasporta» il segnale elettrico relativo ad un bit di informazione o ad un bit di controllo (cioè di comando).

La CPU è l'elemento che gestisce i trasferimenti delle informazioni sul bus. Ogni trasferimento di informazione avviene con un «ciclo» del bus. (*Secondo un certo clock*)
I trasferimenti sono detti di lettura e di scrittura.

- **lettura:**
  la CPU genera indirizzi e segnali di controllo ed è la destinazione dell'informazione

- **scrittura:**
  la CPU genera indirizzi e segnali di controllo ed è la sorgente dell'informazione

Bus di sistema - 2

Bus: Trasferimenti tra memoria e CPU

lettura da memoria (memoria ⇒ CPU)
scrittura in memoria (CPU ⇒ memoria)

Affinché il trasferimento avvenga, è necessario che sul bus siano presenti
- l'indirizzo della parola di memoria interessata
- i segnali di controllo relativi al tipo di trasferimento
- il dato letto da (da scrivere in) memoria

Bus: Trasferimenti tra porte di I/O e CPU

lettura da unitàperiferica (porta ⇒ CPU)
scrittura verso la periferica (CPU ⇒ porta)

Affinché il trasferimento avvenga, è necessario che sul bus siano presenti
- l'indirizzo della porta (registro) della periferica interessata
- i segnali di controllo relativi al tipo di trasferimento
- il dato letto dalla (da scrivere sulla) porta

Segnali di controllo necessari (per lo schema semplificato):

- **WM** (Write Memory): assume il valore 1 per abilitare un'operazione di scrittura in memoria
- **RM** (Read Memory): assume il valore 1 per abilitare un'operazione di lettura in memoria
- **WO** (Write Output): assume il valore 1 per abilitare un'operazione di scrittura su porta di uscita
- **RI** (Read Input): assume il valore 1 per abilitare un'operazione di lettura da porta di ingresso

Partizione del bus

L'insieme dei conduttori che costituiscono il bus può essere considerato suddiviso in **3 sottoinsiemi,** ognuno dedicato a funzioni specifiche e costituito da un numero adeguato di bit, come descritto nel seguito.

Bus indirizzi: presenta la configurazione binaria dell'indirizzo della parola di memoria o della porta cui accedere.

L'indirizzo è sempre fornito dalla CPU che utilizza un suo registro interno d'appoggio (***registro indirizzi***) per presentare la configurazione sul bus. Se sono disponibili k bit (conduttori) di indirizzo, le celle indirizzabili sono 2^k. In genere k = 12. . 32.

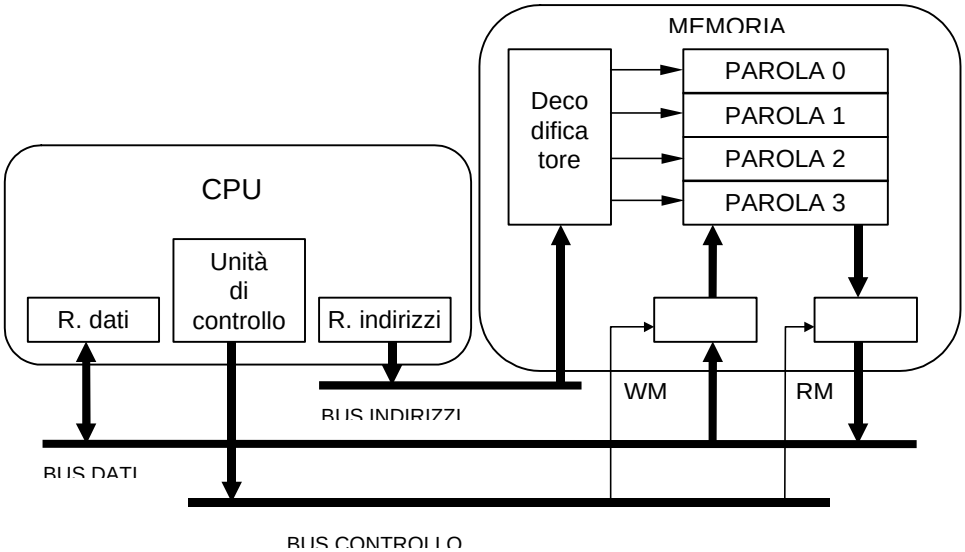
Bus dati: presenta la codifica binaria dell'informazione da trasferire.

Il contenuto può essere fornito dalla memoria o da una porta di ingresso (ciclo di lettura), oppure dalla CPU (ciclo di scrittura). Se sono disponibili h bit (conduttori) per il bus dati è possibile trasferire il contenuto di una parola (registro) di h bit in un unico ciclo. Generalmente h = 8, 16, 32.

Al termine del trasferimento, in caso di lettura, il contenuto del bus dati è memorizzato in un registro d'appoggio della CPU (***registro dati***). In caso di scrittura, la CPU utilizza il registro dati per presentare la configurazione sul bus.

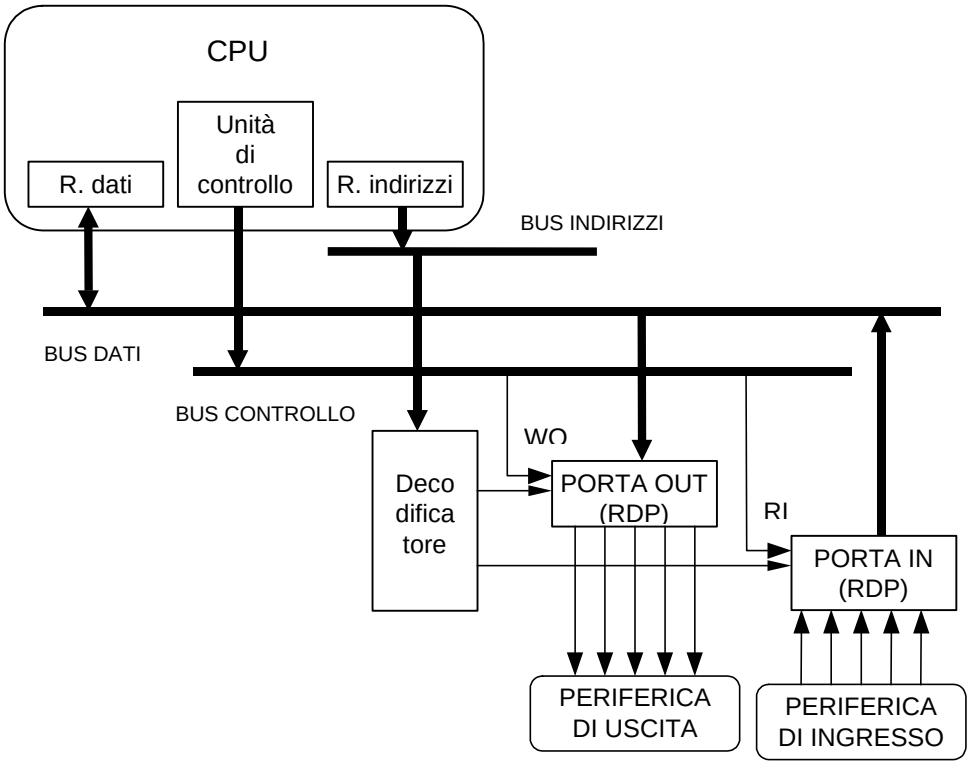
Bus di controllo: presenta i segnali forniti dall'***unità di controllo*** (che fa parte della CPU) che abilitano i circuiti ad effettuare il corretto trasferimento del contenuto del bus dati.

Accesso a memoria



WM è il segnale Write Memory = scrittura memoria
RM è il segnale Read Memory = lettura memoria

Accesso a unità periferica



WO è il segnale Write Output = scrittura su porta
RI è il segnale Read Input = lettura da porta

La memoria di massa -1

La memoria di massa utilizza generalmente un **supporto magnetico** per la memorizzazione delle informazioni.
L'informazione binaria (bit) è memorizzata tramite i due stati di magnetizzazione. E' quindi una memoria **permanente**.

Le tipiche memorie di massa di un calcolatore sono i **dischi magnetici**.

- I dischi rigidi (*hard disk*) hanno tempi di accesso alle informazioni dell'ordine delle decine di millisecondi e le capacità dell'ordine delle centinaia di Megabyte.
- I *floppy disk* hanno tempi di accesso alle informazioni dell'ordine delle centinaia di millisecondi e le capacità dell'ordine del Megabyte.

I **dischi** sono fisicamente suddivisi in tracce e settori, e la lettura/scrittura fisica delle informazioni avviene posizionando la testina di lettura/scrittura sulla traccia e settore voluto.
L'interfaccia di un disco (*drive*) è un dispositivo complesso che risolve i problemi di accesso fisico al disco.

La memoria di massa -2

Le informazioni su disco sono *logicamente* organizzate in **file**.
Un file rappresenta l'unità logica di informazione su disco. Il **Sistema Operativo** fornisce le funzioni e le operazioni necessarie all'accesso ai file. Per utilizzare le informazioni su file, è necessario che queste siano ricopiate nella memoria di lavoro.

I file possono contenere informazioni qualsiasi:
documenti (file di testo), programmi in codice sorgente (file di testo), programmi in codice macchina (file binario eseguibile), immagini (ad esempio mappa di bit).

Altri tipi di memorie di massa

I **nastri magnetici** sono memorie di massa con tempi di accesso più lunghi e capacità di memorizzazione superiori ai dischi e di solito sono utilizzati per memorizzare informazioni «storiche» o di backup e comunque di uso molto poco frequente. Sono estraibili e quindi archiviabili.

I **CD-ROM** sono memorie di massa a sola lettura.
La memorizzazione delle informazioni, e la lettura, utilizzano i principi di funzionamento dei **dischi ottici**.

Struttura della CPU - 1

Caratteristiche di un programma in codice macchina che derivano dall'architettura tipica dei calcolatori, e in particolare dalla struttura della CPU presentata nelle pagine seguenti.

Relativamente alle **istruzioni**:

- le istruzioni di ogni programma sono caricate in parole di memoria contigue.
- l'indirizzo della parola di memoria che contiene la prima istruzione da eseguire all'accensione è noto alla CPU.
- l'inizio dei programmi applicativi è noto al Sistema Operativo.

  Consideriamo il caso semplice in cui ogni istruzione in linguaggio macchina occupa una sola parola di memoria.

Relativamente alle **variabili**:

- sono state riservate le parole di memoria adeguate a contenere i valori delle variabili (dati).
- lo spazio di memoria riservato per le variabili è «separato» da quello che contiene le istruzioni.

  Consideriamo il caso semplice in cui ogni variabile occupa una sola parola di memoria.
- in linguaggio macchina le variabili sono identificate in modo univoco dall'indirizzo corrispondente.
- il nome simbolico di una variabile può essere considerato rappresentativo dell'indirizzo della parola di memoria riservata ad essa.
- il contenuto della parola di memoria riservata ad una variabile rappresenta il valore della variabile.
- Il valore delle variabili sarà determinato dall'esecuzione (operazioni di lettura e assegnamento).

Struttura della CPU - 2

Poiché l'esecuzione **è sequenziale**, è prelevata da memoria un'istruzione alla volta. E' presente un registro (**Program Counter** - **PC**) che viene man mano incrementato in modo da contenere l'indirizzo della parola di memoria in cui è presente la prossima istruzione da eseguire.

L'istruzione da eseguire, una volta prelevata dalla memoria con un'operazione di lettura, è memorizzata nel **registro istruzione** e decodificata (interpretata) dall'**unità di controllo**.

L'**esecuzione** di un'istruzione comporta l'esecuzione di un insieme di passi elementari. Questi possono comportare la generazione di segnali del bus di controllo per l'accesso a memoria o a porta, e la generazione di segnali di controllo interni alla CPU per l'esecuzione di operazioni aritmetiche o logiche.

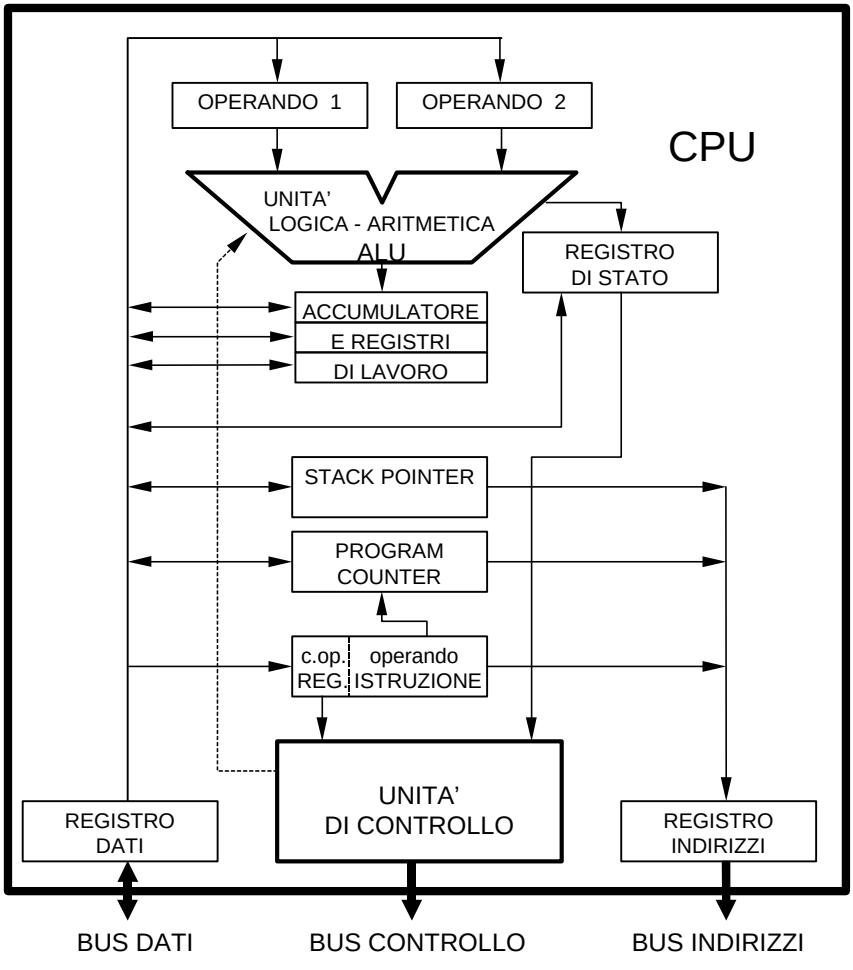
L'**unità di controllo** ha il compito di generare i segnali di controllo che comandano i vari elementi della CPU.

L'**unità aritmetico-logica** (**ALU** - *Arithmetic Logic Unit*) è l'elemento della CPU in grado di eseguire le operazioni.

La CPU è dotata di **registri di lavoro** che mantengono i valori degli operandi che devono essere utilizzati dall'ALU, e i risultati prodotti.

L'unità di controllo genera anche i segnali interni che consentono il trasferimento di informazioni tra i vari registri.

Struttura della CPU (1)



Le frecce indicano i possibili trasferimenti di informazioni
La freccia tratteggiata indica i segnali di controllo per l'ALU

Componenti della CPU (1)

Registro indirizzi: registro di appoggio tra la CPU e il bus indirizzi.

La codifica binaria dell'indirizzo di memoria o della porta di I/O a cui si accede deve essere caricato in questo registro. La sua dimensione (in numero di bit) è pari a quella del bus indirizzi.

Registro dati: registro di appoggio tra la CPU e il bus dati.

In lettura da memoria o da porta, il contenuto del bus dati deve essere memorizzato in questo registro, prima di essere trasferito in un qualsiasi altro elemento della CPU. In scrittura, la codifica binaria da presentare sul bus dati deve essere caricata in questo registro.
La sua dimensione è pari a quella del bus dati.

Program Counter (**PC**): contiene l'indirizzo della prossima istruzione da eseguire.

E' collegato al registro indirizzi per la lettura da memoria dell'istruzione da eseguire. Può ricevere valori da altri registri della CPU.
La sua dimensione è pari a quella del bus indirizzi.

Il **flusso di esecuzione** di un programma in esecuzione è «scandito» dai valori assunti dal PC.
In caso, di esecuzione in sequenza, il PC è incrementato di 1. In caso di salti, nel PC è «forzato» l'indirizzo dell'istruzione destinazione del salto.

Componenti della CPU (2)

Stack Pointer (puntatore alla pila): serve per gestire una porzione della memoria di lavoro, chiamata *stack* (pila).
L'uso delle parole di memoria di quest'area è gestito secondo la modalità LIFO (*Last In First Out*) e cioè l'ultima informazione che è scritta è la prima che viene letta. Lo stack pointer contiene l'indirizzo della prima parola di memoria da leggere nello stack.

L'**inserimento** di un'informazione nello stack comporta:
- incremento dello Stack Pointer
- scrittura dell'informazione all'indirizzo di memoria contenuto nello Stack Pointer

L'**estrazione** di un'informazione dallo stack comporta:
- lettura dell'informazione indirizzata dallo Stack Pointer
- decremento dello Stack Pointer

La possibilitàdi gestire un'area di memoria a stack è di importanza fondamentale per la **chiamata** di un sottoprogramma e per il **ritorno** al programma chiamante.

La **chiamata** di un sottoprogramma comporta, infatti, di
- salvare il valore del Program Counter nello stack. Il valore del PC è l'indirizzo dell'istruzione successiva (indirizzo di ritorno) a quella di chiamata, e quindi è l'istruzione da eseguire al termine dell'esecuzione del sottoprogramma
- forzare l'indirizzo della prima istruzione del sottoprogramma nel Program Counter

Il **ritorno** da sottoprogramma comporta invece di
- estrarre dallo stack e forzare nel Program Counter l'indirizzo di ritorno salvato nello stack alla chiamata

Componenti della CPU (3)

Registro Istruzione: contiene l'istruzione (in linguaggio macchina) correntemente in esecuzione. Il contenuto del Registro Istruzione viene caricato tramite il bus dati ad ogni lettura di istruzione.

In linguaggio macchina il **formato** di un'istruzione è costituito da **2 campi**:

- **codice operativo**: che identifica univocamente il tipo di istruzione
- **operando**: che contiene il riferimento all'operando su cui l'istruzione agisce.

L'operando di un'istruzione può essere un dato, e allora il riferimento è (generalmente) l'indirizzo della parola di memoria riservata per il dato.

L'operando di un'istruzione può essere anche un'altra istruzione (la prossima da eseguire), e allora il riferimento è l'indirizzo della parola di memoria che contiene quest'altra istruzione.

Il campo codice operativo del registro istruzione è un ingresso dell'unità di controllo che, in base alla configurazione di tale campo, interpreta ed esegue l'istruzione corrente, generando gli opportuni segnali di controllo interni ed esterni alla CPU.
Il campo operando (che è in generale un indirizzo) è collegato sia al registro indirizzi che al PC.
La dimensione del registro istruzione è (generalmente) quella del bus dati.

Componenti della CPU (4)

Registri di lavoro: sono dei registri di supporto alle operazioni eseguite all'interno della CPU.
La loro dimensione è quella del bus dati. Il numero di tali registri dipende dal tipo di CPU. Sono identificati da un nome simbolico e referenziabili direttamente dalle istruzioni in linguaggio macchina (o ASSEMBLER).

Registro Indice (I): è uno dei registri di lavoro, e viene utilizzato nel caso di *modalità di indirizzamento a registro indice*.

Accumulatore (A): è uno dei registri di lavoro e, per convenzione, è quello in cui viene memorizzato il risultato di ogni operazione eseguita dall'unità aritmetico-logica.

Unità aritmetico-logica (ALU): esegue le operazioni aritmetiche e logiche elementari.
Le operazioni eseguite dipendono dalla complessità dell'ALU. Le due operazioni fondamentali sono somma e sottrazione, alle quali possono ricondursi operazioni più complesse quali, moltiplicazione, divisione, confronto, ecc..
L'unità aritmetico logica è a 2 operandi in ingresso e fornisce 1 risultato in uscita. I valori degli operandi sono presentati all'ALU tramite i registri di appoggio e il risultato viene memorizzato generalmente nell'accumulatore. I diversi tipi di operazioni vengono abilitati dall'unità di controllo tramite opportuni segnali.

Registri di appoggio (non referenziabili dalle istruzioni in linguaggio macchina o ASSEMBLER):

- per l'unità aritmetico logica (*operando1* e *operando2*): necessari per presentare all'ALU gli operandi su cui eseguire l'operazione.
- per il *calcolo degli indirizzi*: necessario per consentire il calcolo dell'indirizzo dell'operando in presenza delle diverse modalità di indirizzamento.

Componenti della CPU (5)

Registro di stato: è un registro che raggruppa dei bit che hanno significato singolarmente per rappresentare delle condizioni.

Ogni bit riporta indicazioni relative all'esito dell'operazione aritmetica o logica eseguita dall'ALU (viene quindi «scritto» dall'ALU al termine di ogni operazione eseguita). I bit di stato (*flag*) più significativi sono:

carry:
: viene posto a 1 quando l'operazione aritmetica di somma (o sottrazione) tra due operandi genera riporto (o prestito)

zero:
: viene posto a 1 quando il risultato dell'operazione eseguita dall'ALU vale zero (e quindi il contenuto dell'accumulatore è zero)

segno:
: viene posto a 1 quando il risultato dell'operazione eseguita dall'ALU è negativo (il bit più significativo del risultato è 1)

overflow:
: viene posto a 1 quando, dopo una somma o sottrazione tra interi in complemento a 2, il risultato è di segno discorde rispetto a quello concorde degli operandi (operandi di segno discorde non possono dare overflow). Questo indica che il valore numerico ottenuto non è rappresentabile con il numero di bit a disposizione.

I bit del registro di stato vengono interpretati dall'unità di controllo nell'esecuzione di *istruzioni di salto condizionato*.

Componenti della CPU (6)

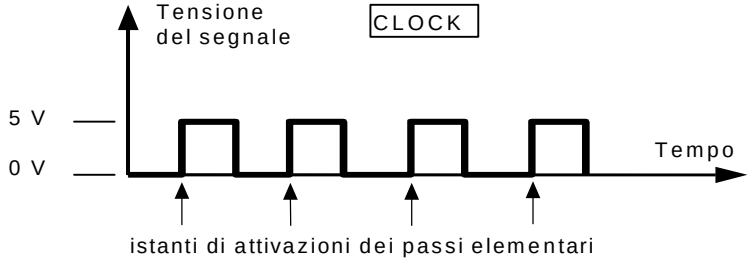
Unità di controllo: genera la sequenza di passi elementari necessari all'esecuzione di ogni specifica istruzione macchina.

Ogni passo elementare è composto da una serie di comandi elementari che possono essere esterni (segnali sul bus di controllo) o interni (per l'ALU e per i trasferimenti tra registri).

La sequenza di passi elementari che viene attivata ciclicamente dall'unità di controllo può essere schematizzata in tre fasi distinte:

- **fase di fetch**:
  Program Counter usato per fornire l'indirizzo di lettura
  acquisizione da memoria dell'istruzione da eseguire
  Program Counter incrementato, pronto per l'istruzione seguente
- **fase di decodifica:**
  interpretazione del codice operativo dell'istruzione
- **fase di esecuzione**:
  a seconda del codice operativo, l'unità di controllo attiva la fase di esecuzione pertinente all'istruzione in esecuzione

Il ritmo temporale dei vari passi è imposto dal **segnale di clock.**



CPU: Unita` di Controllo

Fetch/execute

- L'*unita` di controllo* ha il compito di:
  - reperire dalla memoria centrale le istruzioni di un programma (**fetch**);
  - interpretarle;
  - farle eseguire (**execute**).

- La CPU si comporta quindi come segue:

ripeti
FETCH di una istruzione
EXECUTE dell' istruzione
finche` istruzione = HALT oppure ERRORE

- ***FETCH*** : provoca il trasferimento e la decodifica della prossima istruzione da eseguire (il cui indirizzo e` nel registro PC). Le istruzioni sono organizzate in memoria in sequenza.

- ***EXECUTE***: l'elaboratore esegue l'istruzione trovata (che` e` stata caricata nel registro IR). Istruzioni particolari possono alterare il flusso sequenziale (salto, chiamata di sotto-programmi).

Linguaggio macchina - 1

Ogni tipo di CPU ha un suo proprio linguaggio macchina (*set di istruzioni*) direttamente interpretabile ed eseguibile. Ogni programma, per essere eseguito da una CPU deve quindi essere disponibile (eventualmente tradotto) nel linguaggio macchina **specifico del tipo di CPU.**

Ogni istruzione in linguaggio macchina è rappresentata da bit e quindi è costituita da una sequenza di «0» e «1». Ogni istruzione è costituita logicamente da due campi:

- il campo **codice operativo** è costituito dalla configurazione binaria che **identifica univocamente il tipo di istruzione**
- il campo **operando** contiene la configurazione binaria che **rappresenta un indirizzo di memoria**

Classi di istruzioni tipiche in linguaggio macchina:

1. istruzioni di trasferimento dati da e in memoria
2. istruzioni di trasferimento dati da e in periferica
3. istruzioni aritmetico-logiche
4. istruzioni di modifica del flusso di esecuzione
5. istruzioni ausiliarie

Consideriamo un *set di istruzioni* semplificato e congruente con la struttura della CPU, della memoria e delle interfacce di I/O considerata fino a ora.

Per motivi di chiarezza, rappresenteremo il codice operativo di ogni istruzione anche con una sigla *mnemonica* (ASSEMBLER).

Linguaggio macchina –2

cod. operativo	operando	significato
0000 (LDA)	*indirizzo* operando	mem(*indirizzo*) → ACC
0001 (STA)	*indirizzo* operando	ACC → mem(*indirizzo*)
0010 (ADD)	*indirizzo* operando	ACC + mem(*indirizzo*) → ACC
0011 (SUB)	*indirizzo* operando	ACC - mem(*indirizzo*) → ACC
0100 (JMP)	*indirizzo* istruzione	istruzione di salto incondizionato *indirizzo* → PC (salta a *indirizzo*)
0101 (JZ)	*indirizzo* istruzione	istruzione di salto condizionato se bit di stato $_{zero}$ =1 *indirizzo* → PC
0110 (IN)	*indirizzo* periferica	porta (*indirizzo*) → ACC
0111 (OUT)	*indirizzo* periferica	ACC → porta(*indirizzo*)
1000 (CALL)	*indirizzo* istruzione	chiamata a sottoprogramma PC → stack(Stack Pointer) *indirizzo* → PC
1001 (RET)		ritorno da sottoprogramma stack(Stack Pointer) → PC
1010 (LDI)	*indirizzo* operando	mem(*indirizzo*) → I
(J*cond*) 1011 (JC) 1100 (JS) 1101 (JO)	*indirizzo* istruzione	istruzione di salto condizionato dove *cond* è uno dei bit di stato (C=carry, S=segno, O=overflow) se bit di stato $_{cond}$ =1 *indirizzo* → PC

mem(*indirizzo*):	parola di memoria indirizzata da *indirizzo*
porta (*indirizzo*):	porta indirizzata da *indirizzo*
stack(Stack Pointer):	parola di memoria dello stack indirizzata dallo Stack Pointer
ACC:	Accumulatore
I:	Registro Indice

Ritorna

Prodotto per somme ripetute
in linguaggio macchina (forma simbolica)

indirizzo di memoria	descrizione simbolica del contenuto della parola di memoria	programma in C
129	.....	
130	IN 1	{ leggi w;
131	STA 153	
132	IN 1	leggi y;
133	STA 154	
134	LDA 151	sp = 0;
135	STA 156	
136	LDA 154	ns = y;
137	STA 157	
138	LDA 157	while (ns != 0)
139	JZ 147	
140	LDA 156	{ sp = sp + w;
141	ADD 153	
142	STA 156	
143	LDA 157	ns = ns -1;
144	SUB 152	
145	STA 157	
146	JMP 138	}
147	LDA 156	z = sp;
148	STA 155	
149	OUT 2	scrivi z;
150	JMP 2000	}
151	0000000000000000	const int zero=0;
152	0000000000000001	const int uno=1;
153		int w;
154		int y;
155		int z;
156		int sp;
157		int ns;
158	.....	

gli indirizzi in quest'esempio sono rappresentati in decimale

In quest'esempio:

1 è l'indirizzo della porta di ingresso associata alla tastiera
2 è l'indirizzo della porta di uscita associata al terminale
2000 è l'indirizzo di ritorno a Sistema Operativo

Prodotto per somme ripetute in linguaggio macchina binario

indirizzo di memoria	contenuto della parola di memoria	
000010000001	.....	....
000010000010	0110	000000000001
000010000011	0001	000010011001
000010000100	0110	000000000001
000010000101	0001	000010011010
000010000110	0000	000010010111
000010000111	0001	000010011100
000010001000	0000	000010011010
000010001001	0001	000010011101
000010001010	0000	000010011101
000010001011	0101	000010010011
000010001100	0000	000010011100
000010001101	0010	000010011001
000010001110	0001	000010011100
000010001111	0000	000010011101
000010010000	0011	000010011000
000010010001	0001	000010011101
000010010010	0100	000010001010
000010010011	0000	000010011100
000010010100	0001	000010011011
000010010101	0111	000000000010
000010010110	0100	011111010000
000010010111	0000	000000000000
000010011000	0000	000000000001
000010011001	xxxx	xxxxxxxxxxxx
000010011010	xxxx	xxxxxxxxxxxx
000010011011	xxxx	xxxxxxxxxxxx
000010011100	xxxx	xxxxxxxxxxxx
000010011101	xxxx	xxxxxxxxxxxx
000010011110	....	....

Parole di memoria da 16 bit.
Memoria da 4k parole (12 bit per specificare l'indirizzo)

Passi elementari per l'esecuzione delle istruzioni

- fase di fetch (comune per tutte le istruzioni):
  PC → Registro Indirizzi
  mem(Registro Indirizzi)→Registro Dati
  Registro Dati→Registro Istruzione
  PC+1 → PC

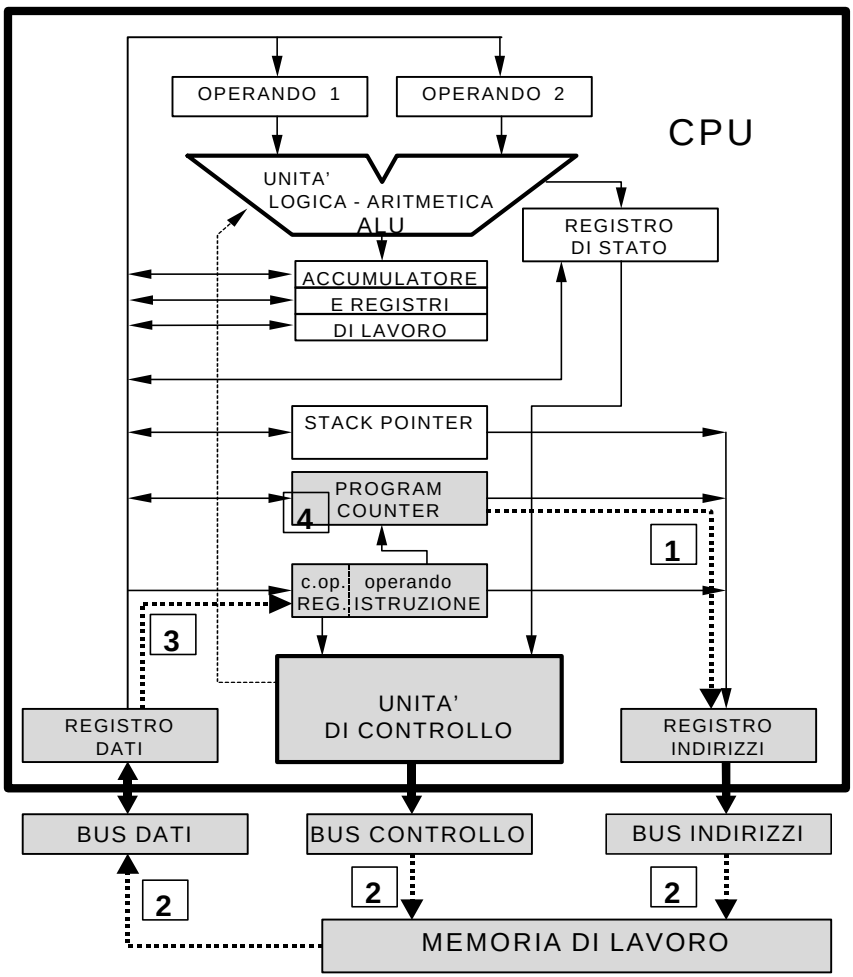
- fase di esecuzione LDA:
  Registro Istruzione$_{operando}$→ Registro Indirizzi
  mem(Registro Indirizzi)→Registro Dati
  Registro Dati→Accumulatore

- fase di esecuzione ADD:
  Accumulatore→ Operando 1
  Registro Istruzione$_{operando}$→ Registro Indirizzi
  mem(Registro Indirizzi)→ Registro Dati
  Registro Dati→ Operando 2
  somma→Accumulatore

- fase di esecuzione JZ:
  se (bit di stato$_{\mathbf{zero}}$= 1)
  Registro Istruzione$_{operando}$→ PC

- fase di esecuzione OUT:
  Registro Istruzione$_{operando}$→ Registro Indirizzi
  Accumulatore→ Registro Dati
  Registro Dati→ porta(Registro Indirizzi)

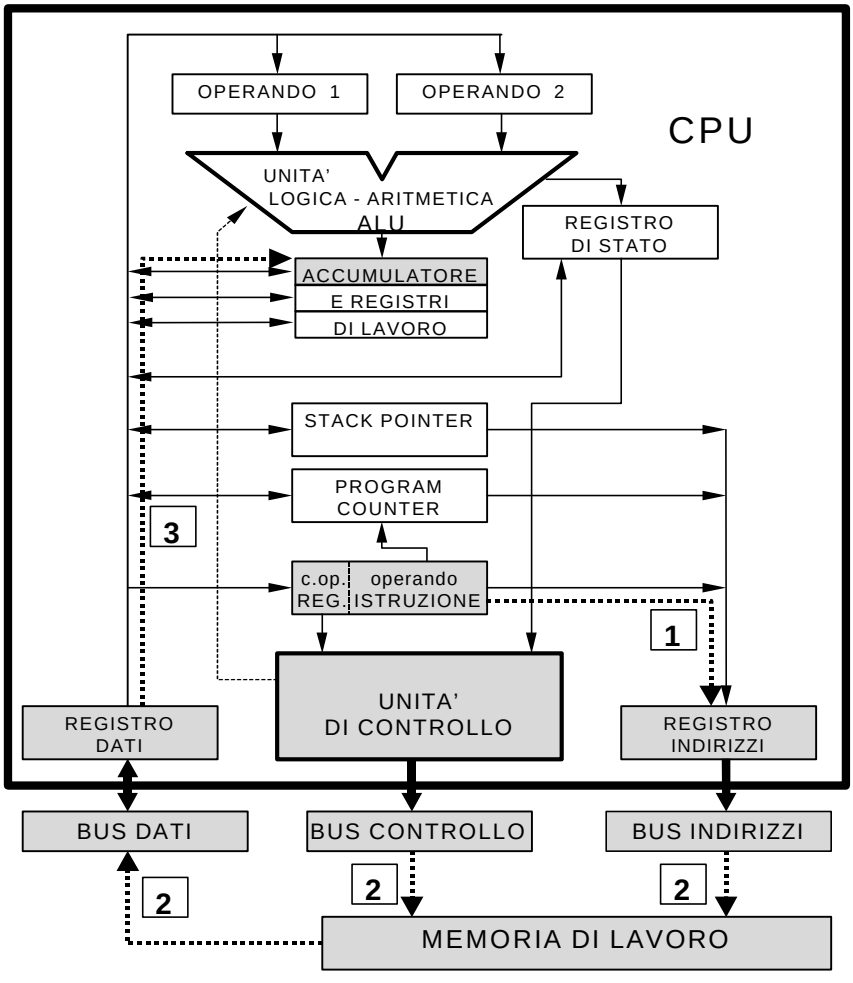
FASE DI FETCH



*

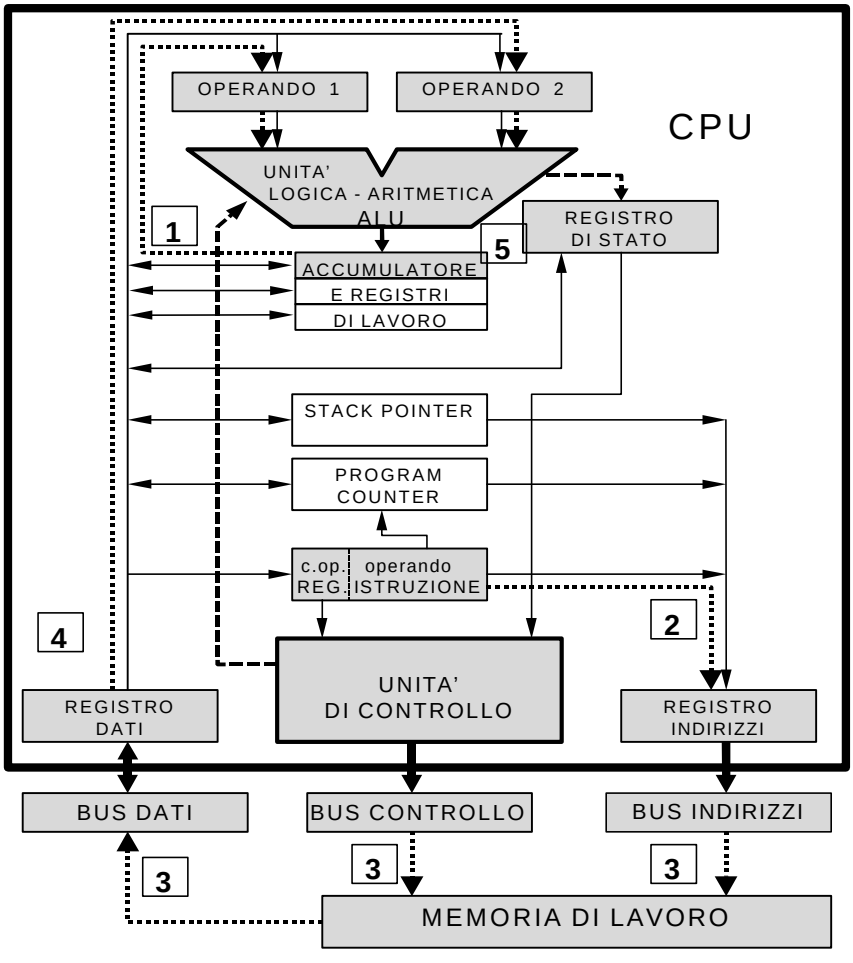
1 - PC → Registro Indirizzi
2 - mem(Registro Indirizzi) → Registro Dati
3 - Registro Dati → Registro Istruzione
4 - PC+1 → PC

Fase di esecuzione LDA



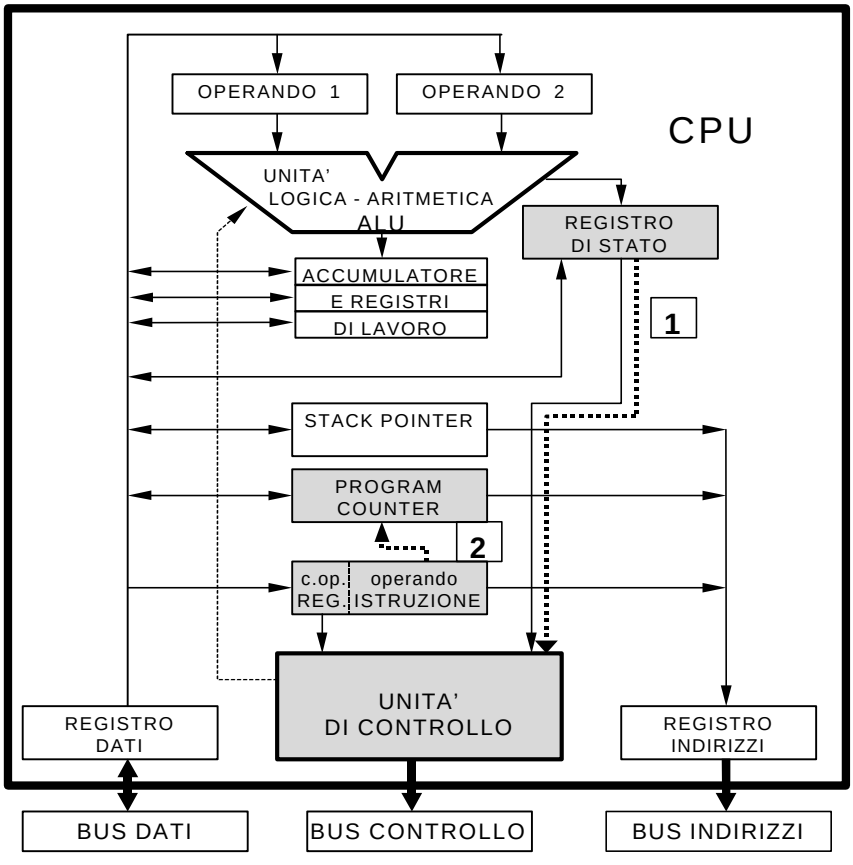
1 - Registro Istruzione$_{operando}$ → Registro Indirizzi
2 - mem(Registro Indirizzi) → Registro Dati
3 - Registro Dati → Accumulatore

Fase di esecuzione ADD



1 - Accumulatore → Operando 1
2 - Registro Istruzione$_{operando}$ → Registro Indirizzi
3 - mem(Registro Indirizzi) → Registro Dati
4 - Registro Dati → Operando 2
5 - *somma* → Accumulatore

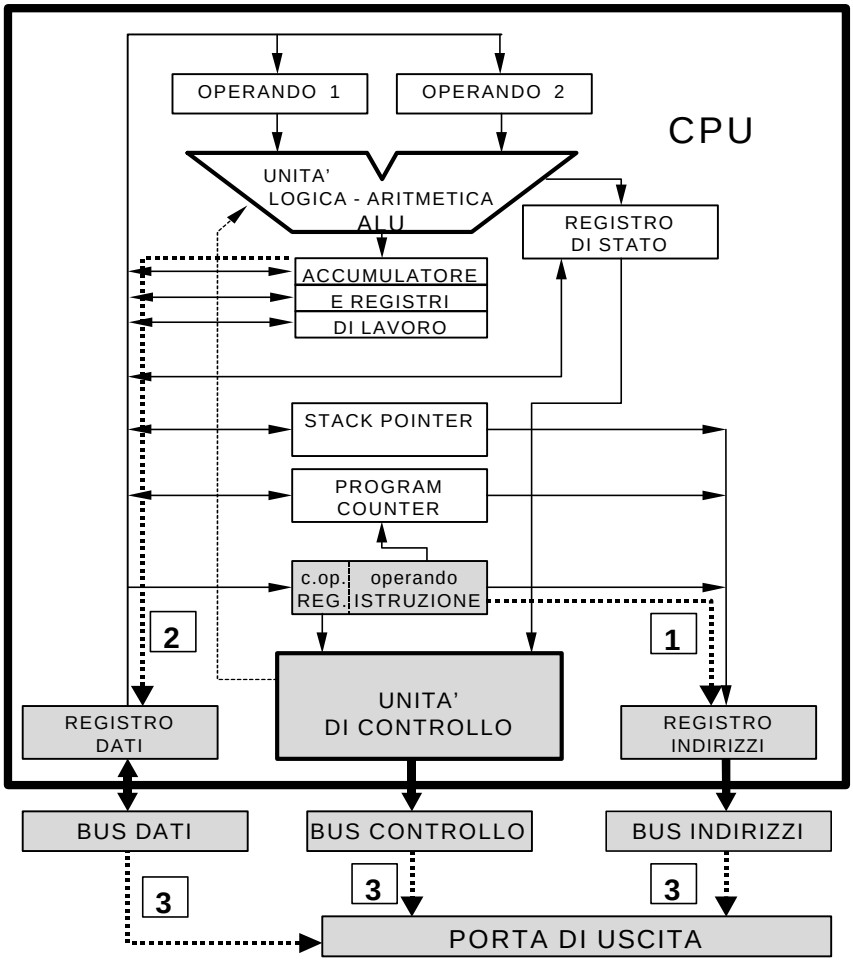
Fase di esecuzione JZ



1 - ***se*** (bit di stato$_{\mathbf{zero}}$= 1)

2 - Registro Istruzione$_{operando}$ → PC

fase di esecuzione OUT



1 - Registro Istruzione$_{operando}$ → Registro Indirizzi
2 - Accumulatore → Registro Dati
3 - Registro Dati → porta(Registro Indirizzi)

Linguaggio ASSEMBLER

E' il **linguaggio *simbolico*** che consente di programmare un calcolatore utilizzando le istruzioni del linguaggio macchina. Per questo motivo viene detto linguaggio di programmazione di basso livello.

In ASSEMBLER:

- ogni istruzione corrisponde a una e una sola istruzione in linguaggio macchina (e viceversa)
- gli indirizzi di memoria sono espressi in modo simbolico utilizzando nomi simbolici che costituiscono delle «etichette» (*label*) associate alle posizioni di memoria. Quindi una variabile o una destinazione di salto sono esprimibili tramite una etichetta
- esistono delle direttive che consentono di riservare lo spazio di memoria adeguato a contenere una variabile di un certo tipo
- esistono delle direttive che consentono di definire dei *simboli* per la sostituzione letterale di valori costanti

Un programma scritto in ASSEMBLER per essere eseguito deve essere **tradotto in linguaggio macchina binario**, in modo tale da tradurre i codici mnemonici delle istruzioni in codici operativi, sostituire tutti i riferimenti simbolici degli indirizzi con la loro forma binaria, e riservare lo spazio di memoria per le variabili.

L'operazione di traduzione viene eseguita da un particolare programma, detto ASSEMBLATORE.

Ai codici numerici

Linguaggio Assembler

istruzione	significato
LDA *indirizzo*	mem(*indirizzo*) → ACC
STA *indirizzo*	ACC → mem(*indirizzo*)
ADD *indirizzo*	ACC + mem(*indirizzo*) → ACC
SUB *indirizzo*	ACC - mem(*indirizzo*) → ACC
JMP *indirizzo*	istruzione di salto incondizionato *indirizzo* → PC (salta a *indirizzo*)
JZ *indirizzo*	istruzione di salto condizionato se bit di stato $_{\mathbf{zero}}$ =1 *indirizzo* → PC
IN *indirizzo*	porta (*indirizzo*) → ACC
OUT *indirizzo*	ACC → porta(*indirizzo*)
CALL *indirizzo*	chiamata a sottoprogramma PC → stack(Stack Pointer) *indirizzo* → PC
RET	ritorno da sottoprogramma stack(Stack Pointer) → PC
LDI *indirizzo*	mem(*indirizzo*) → I
J*cond* indirizzo *JC* *JS* *JO*	istruzione di salto condizionato dove *cond* è uno dei bit di stato (C=carry, S=segno, O=overflow) se bit di stato $_{\mathbf{cond}}$ =1 *indirizzo* → PC

PRODOTTO PER SOMME RIPETUTE IN ASSEMBLER

programma in ASSEMBLER			programma in C
label	**operaz.**	**operando**	
EXIT	**EQU**	**2000**	
PORTA_TASTIERA	**EQU**	**1**	
PORTA_VIDEO	**EQU**	**2**	
	IN	**PORTA_TASTIERA**	`{ leggi w;`
	STA	**W**	
	IN	**PORTA_TASTIERA**	`leggi y;`
	STA	**Y**	
	LDA	**ZERO**	`sp = 0;`
	STA	**SP**	
	LDA	**Y**	`ns = y;`
	STA	**NS**	
WHILE	**LDA**	**NS**	`while (ns != 0)`
	JZ	**ENDWHILE**	
	LDA	**SP**	`{ sp = sp + w;`
	ADD	**W**	
	STA	**SP**	
	LDA	**NS**	`ns = ns -1;`
	SUB	**UNO**	
	STA	**NS**	
	JMP	**WHILE**	`}`
ENDWHILE	**LDA**	**SP**	`z = sp;`
	STA	**Z**	
	OUT	**PORTA_VIDEO**	`scrivi z;`
	JMP	**EXIT**	`}`
ZERO	**DW**	**0**	`const int zero=0;`
UNO	**DW**	**1**	`const int uno=1;`
W	**DW**	**?**	`int w;`
Y	**DW**	**?**	`int y;`
Z	**DW**	**?**	`int z;`
SP	**DW**	**?**	`int sp;`
NS	**DW**	**?**	`int ns;`

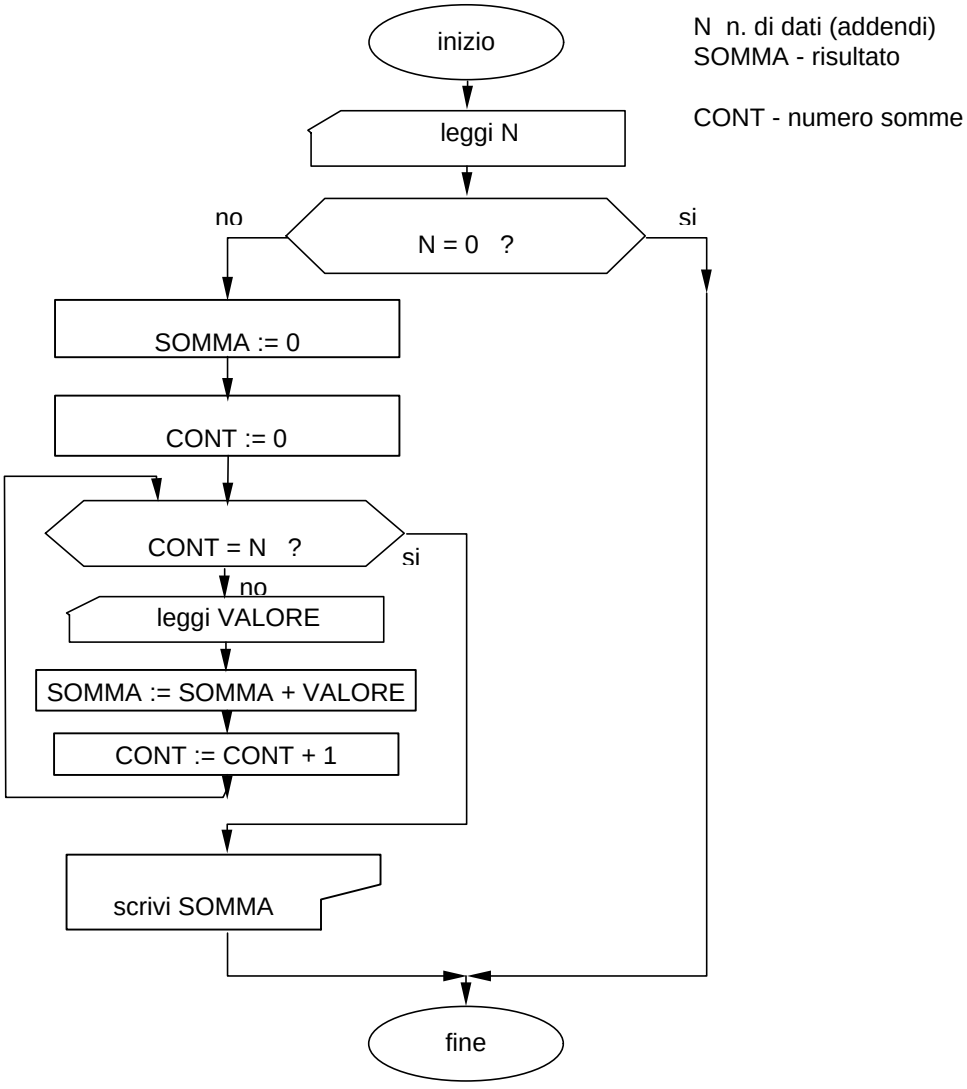
`label EQU valore` **label assume il ruolo di costante simbolica**

`label DW valore` **`label DW ?`**

(**DW** = Define Word): direttiva all'assemblatore per riservare in quella posizione una parola di memoria per la variabile **`label`** eventualmente inizializzata a **`valore`**

ESEMPIO 1- SOMMA N VALORI INTERI - SCHEMA A BLOCCHI

Si vuole calcolare la somma di N valori interi forniti dall'utente. Il numero di valori da acquisire (N) è fornito dall'utente.



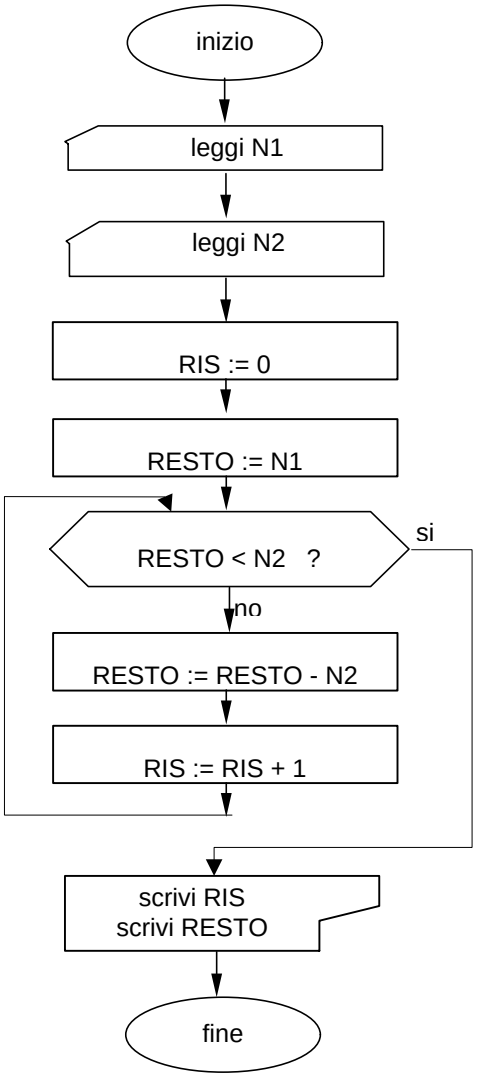
ESEMPIO 1- SOMMA N VALORI INTERI - PROGRAMMA IN ASSEMBLER

Si vuole calcolare la somma di N valori interi forniti dall'utente. Il numero di valori da acquisire (N) è fornito dall'utente.

```
                EXIT              EQU  2000
                PORTA_TASTIERA    EQU  1
                PORTA_VIDEO       EQU  2

START           IN   PORTA_TASTIERA
                STA  N
                JZ   FINE
                LDA  ZERO
                STA  SOMMA
                LDA  ZERO
                STA  CONT
CICLO           LDA  CONT
                SUB  N
                JZ   VISUALIZZA
                IN   PORTA_TASTIERA
                STA  VALORE
                LDA  SOMMA
                ADD  VALORE
                STA  SOMMA
                LDA  CONT
                ADD  UNO
                STA  CONT
                JMP  CICLO
VISUALIZZA      LDA  SOMMA
                OUT  PORTA_VIDEO
FINE            JMP  EXIT
ZERO            DW   0
UNO             DW   1
N               DW   ?
CONT            DW   ?
SOMMA           DW   ?
VALORE          DW   ?
```

ESEMPIO 2- DIVISIONE TRA INTERI - SCHEMA A BLOCCHI

La divisione N1/N2 viene effettuata, con quest'algoritmo, per sottrazioni successive, con l'ipotesi N2 > 0 e N1≥0. L'algoritmo calcola il risultato RIS e il resto RESTO della divisione intera.



ESEMPIO 2- DIVISIONE TRA INTERI - PROGRAMMA IN ASSEMBLER

La divisione N1/N2 viene effettuata, con quest'algoritmo, per sottrazioni successive, con l'ipotesi N2 > 0 e N1≥0. L'algoritmo calcola il risultato e il resto della divisione intera.

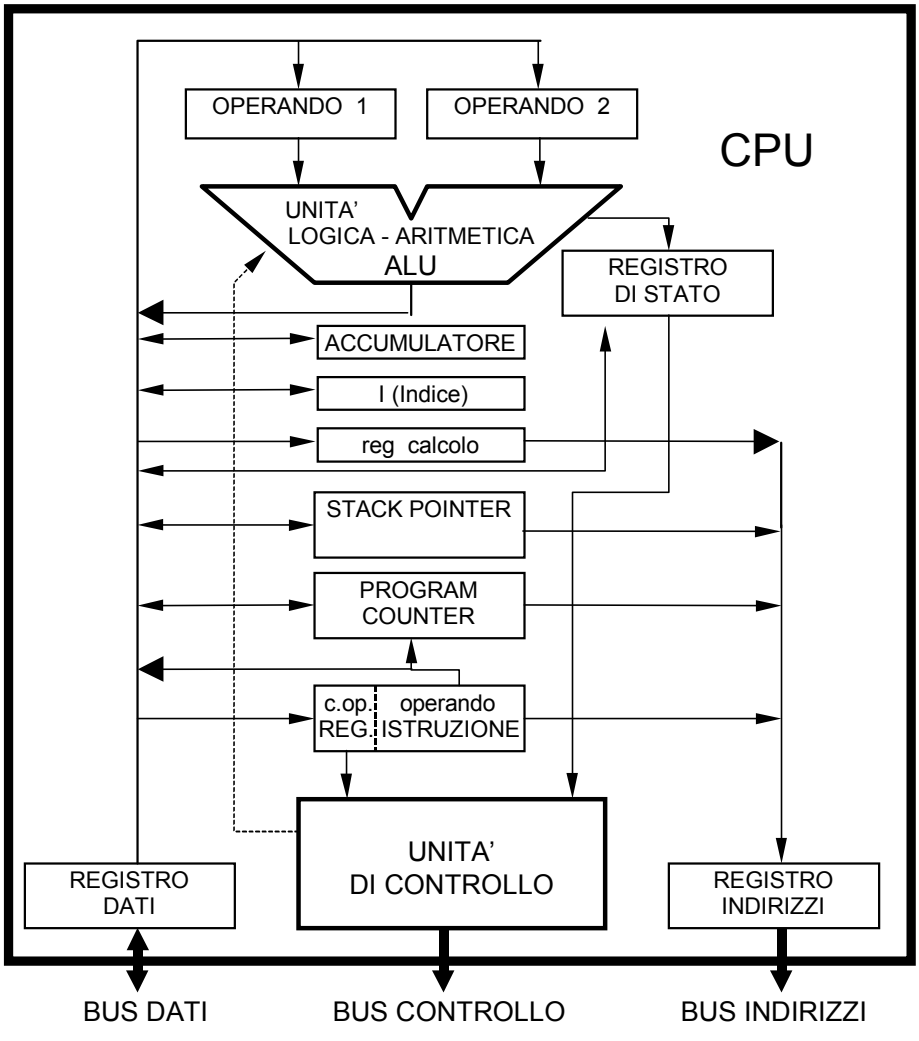
```
                EXIT                EQU  2000
                PORTA_TASTIERA      EQU  1
                PORTA_VIDEO         EQU  2

START     IN    PORTA_TASTIERA
                STA  N1
                IN   PORTA_TASTIERA
                STA  N2
                LDA  ZERO
                STA  RIS
                LDA  N1
                STA  RESTO
CICLO     LDA   RESTO
                SUB  N2
                JS   SCRIVI
                LDA  RESTO
                SUB  N2
                STA  RESTO
                LDA  RIS
                ADD  UNO
                STA  RIS
                JMP  CICLO
SCRIVI          LDA  RIS
                OUT  PORTA_VIDEO
                LDA  RESTO
                OUT  PORTA_VIDEO
FINE            JMP  EXIT
ZERO            DW   0
UNO             DW   1
N1              DW   ?
N2              DW   ?
RIS             DW   ?
RESTO     DW    ?
```

Memo Linguaggio Assembler

istruzione	significato
LDA *indirizzo*	Carica da memoria a registro ACC
STA *indirizzo*	Memorizza da registro ACC
ADD *indirizzo*	Somma da memoria a registro ACC
SUB *indirizzo*	Sottrai da memoria a registro ACC
JMP *indirizzo*	Salta incondizionatamente all'indirizzo)
JZ *indirizzo*	Salto condizionato se bit di stato **zero** =1
IN *indirizzo*	Leggi da porta di input *indirizzo*
OUT *indirizzo*	Scrivi a porta di output *indirizzo*
CALL *indirizzo*	chiamata a sottoprogramma
RET	ritorno da sottoprogramma
LDI *indirizzo*	Carica da memoria il registro I
J*cond* indirizzo *JC, JS, JO*	istruzione di salto condizionato (C=carry, S=segno, O=overflow) se bit di stato ***cond*** =1

`label EQU valore`
`label DW valore`
(**`DW`** = Define Word): direttiva all'assemblatore per riservare in quella posizione una parola di memoria per la variabile **`label`** eventualmente inizializzata a **`valore`**
`(se non c'è un valore si mette '?'`

label assume il ruolo di costante simbolica
`label DW ?`



MODALITÀ DI INDIRIZZAMENTO
E FORMATO ISTRUZIONI MACCHINA

Formato istruzione:

Codice Operativo	Operando
	indirizzo

parola di memoria

Codice Operativo: m bit (2^m istruzioni distinte)
Campo Operando: n bit (2^n indirizzi distinti)

Il linguaggio macchina, e quindi il linguaggio ASSEMBLER, prevede diverse **modalità di indirizzamento** (oltre a quello diretto) che rendono il set istruzioni più potente e consentono maggior flessibilità ed efficienza.

La modalità di indirizzamento è un campo aggiuntivo nel formato istruzione e definisce il modo in cui deve essere utilizzato il valore presente nel Campo Operando per calcolare l'indirizzo dell'operando stesso.

FORMATO ISTRUZIONE:

Codice Operativo	Mod. Indir.	Operando
		indirizzo

parola di memoria

Codice Operativo: m bit (2^m istruzioni distinte)
Modalità di Indirizzamento: s bit (2^s modalità distinte)
Campo Operando: n bit (2^n indirizzi distinti)

Non tutte le istruzioni ASSEMBLER ammettono tutte le modalità di indirizzamento.

MODALITÀ DI INDIRIZZAMENTO (1)

INDIRIZZAMENTO DIRETTO:

- il Campo Operando contiene l'indirizzo dell'operando stesso
- un accesso a memoria per accedere all'operando

in ASSEMBLER: ETICHETTA

Ad esempio:
LDA X significato mem(X)→ A

INDIRIZZAMENTO INDIRETTO: Puntatore

- il Campo Operando contiene l'indirizzo di memoria di una parola che contiene l'indirizzo dell'operando
- due accessi a memoria per accedere all'operando

in ASSEMBLER: @ETICHETTA

Ad esempio:
LDA @X significato mem(mem(X))→ A

MODALITÀ DI INDIRIZZAMENTO (2)

INDIRIZZAMENTO RELATIVO A REGISTRO INDICE: Indice di array

- il Campo Operando contiene un indirizzo di memoria che viene sommato al contenuto del registro indice (I) per ottenere l'indirizzo dell'operando

Questa modalitàconsente di indirizzare in modo parametrico una sequenza di parole di memoria contigue (Campo Operando = indirizzo base e valore di I = spiazzamento)

in ASSEMBLER: ETICHETTA(I)

Ad esempio:
LDA X(I) significato mem(X+I) → A

INDIRIZZAMENTO IMMEDIATO:

- il Campo Operando contiene il **valore** dell'operando (il valore può essere espresso in modo esplicito oppure simbolico)

in ASSEMBLER: #ETICHETTA (o valore)

Ad esempio:
LDA #X significato X → A

Politecnico di Milano - Anno Accademico 2000-2001 - Informatica B Meccanici-Energetici
2^ Prova in Itinere - 12 Febbraio 2001 – Elaborato B

Es 7 - Tradurre in assembler i seguenti frammenti di programma C (usare gli stessi identificatori di variabile).

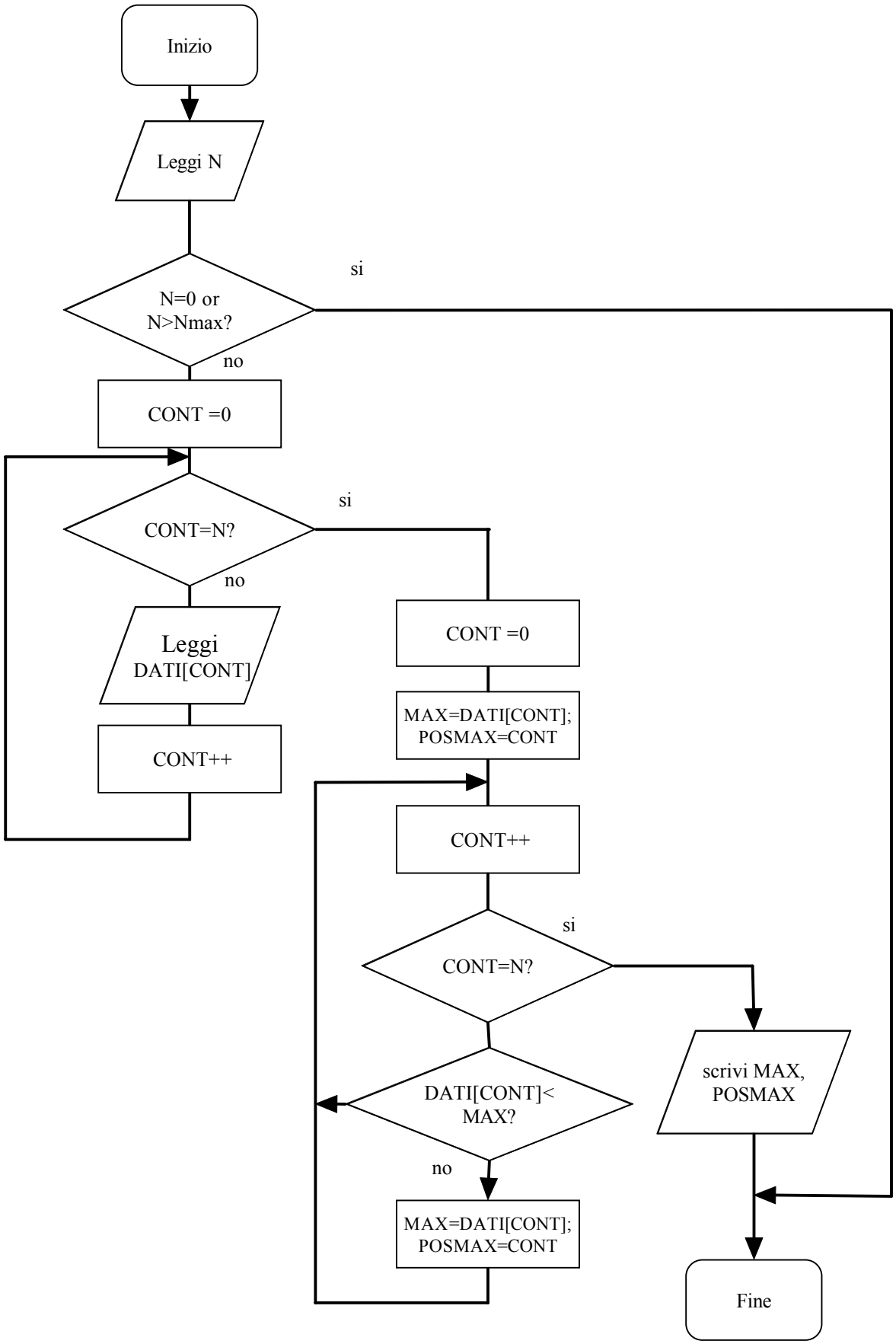
	LDI NEXT	SCORE[NEXT]= MAX-1;
	LDA MAX	
	SUB #1	
	STA SCORE(I)	
	LDA NEXT	++NEXT;
	ADD #1	
	STA NEXT	
	LDA @PNEXT	*PNEXT -= 1;
	SUB #1	
	STA @PNEXT	
NMAX	EQU 5	#define NMAX 5
PNEXT	DW ?	int *PNEXT;
NEXT	DW ?	int NEXT;
MAX	DW ?	int MAX;
SCORE	DW ?	int SCORE [NMAX];
	DW ?	
	DW ?	
	DW ?	
	DW ?	

ESEMPIO 3A- CALCOLO DEL MASSIMO IN UN ARRAY DI INTERI

Si vuole acquisire una sequenza di N (≤ Nmax) valori interi, memorizzarli in un array, e trovare il valore massimo e la sua posizione. Si ipotizzi che i valori acquisiti siano ≥0.

```
                EXIT                EQU  2000
                PORTA_TASTIERA      EQU  1
                PORTA_VIDEO         EQU  2
                ZERO                EQU  0
                UNO                 EQU  1
                NMAX                EQU  10

START     IN    PORTA_TASTIERA
                STA  N
                JZ   FINE
                LDA  #NMAX
                SUB  N
                JS   FINE
                LDA  #ZERO
                STA  CONT
                LDI  CONT
ACQUISISCI      LDA  CONT
                SUB  N
                JZ   CALCOLA
                IN   PORTA_TASTIERA
                STA  DATI(I)
                LDA  CONT
                ADD  #UNO
                STA  CONT
                LDI  CONT
                JMP  ACQUISISCI
CALCOLA         LDA  #ZERO
                STA  CONT
                LDI  CONT
                LDA  DATI(I)
                STA  MAX
                LDA  CONT
                STA  POSMAX


CICLO           LDA  CONT
                ADD  #UNO
                STA  CONT
                LDA  CONT
                SUB  N
                JZ   SCRIVI
                LDI  CONT
```

```
                LDA   DATI(I)
                SUB   MAX
                JS    CICLO
                LDA   DATI(I)
                STA   MAX
                LDA   CONT
                STA   POSMAX
                JMP   CICLO
SCRIVI          LDA   MAX
                OUT   PORTA_VIDEO
                LDA   POSMAX
                OUT   PORTA_VIDEO
FINE            JMP   EXIT
N               DW    ?
CONT            DW    ?
MAX             DW    ?
POSMAX          DW    ?
DATI            DW    ?
                DW    ?
                DW    ?
                DW    ?
                DW    ?
                DW    ?
                DW    ?
                DW    ?
                DW    ?
                DW    ?
```


Inizio
Leggi N
N=0 or N>Nmax?
si
no
CONT =0
CONT=N?
si
no
Leggi DATI[CONT]
CONT++
CONT =0
MAX=DATI[CONT]; POSMAX=CONT
CONT++
CONT=N?
si
DATI[CONT]< MAX?
no
MAX=DATI[CONT]; POSMAX=CONT
scrivi MAX, POSMAX
Fine

ESEMPIO 3B- CALCOLO DEL MASSIMO IN UN ARRAY DI INTERI

Versione con chiamata a sottoprogramma per l'aggiornamento del massimo e della sua posizione.

```
                EXIT                EQU  2000
                PORTA_TASTIERA      EQU  1
                PORTA_VIDEO         EQU  2
                ZERO                EQU  0
                UNO                 EQU  1
                NMAX                EQU  10

START           IN   PORTA_TASTIERA
                STA  N
                JZ   FINE
                LDA  #NMAX
                SUB  N
                JS   FINE
                LDA  #ZERO
                STA  CONT
ACQUISISCILDA   CONT
                SUB  N
                JZ   CALCOLA
                IN   PORTA_TASTIERA
                LDI  CONT
                STA  DATI(I)
                LDA  CONT
                ADD  #UNO
                STA  CONT
                JMP  ACQUISISCI
CALCOLA         LDA  #ZERO
                STA  CONT
                LDI  CONT
                CALL AGGIORNA
CICLO           LDA  CONT
                ADD  #UNO
                STA  CONT
                SUB  N
                JZ   SCRIVI
```

```
                LDI  CONT
                LDA  DATI(I)
                SUB  MAX
                JS   CICLO
                CALL AGGIORNA
                JMP  CICLO
SCRIVI          LDA  MAX
                OUT  PORTA_VIDEO
                LDA  POSMAX
                OUT  PORTA_VIDEO
FINE            JMP  EXIT
AGGIORNA        LDA  DATI(I)
                STA  MAX
                LDA  CONT
                STA  POSMAX
                RET
N               DW   ?
CONT            DW   ?
MAX             DW   ?
POSMAX          DW   ?
DATI            DW   ?
                DW   ?
                DW   ?
                DW   ?
                DW   ?
                DW   ?
                DW   ?
                DW   ?
                DW   ?
                DW   ?
```