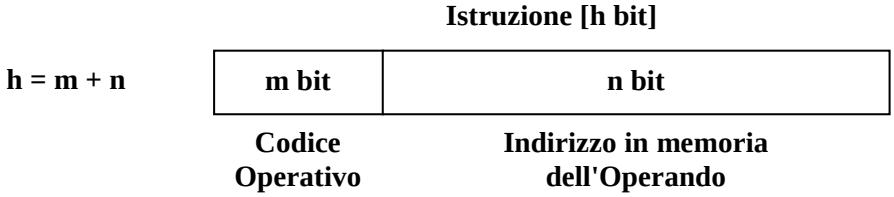
MACCHINA DI VON NEUMANN [1]

- COSA FAREMO
  - Modello esecuzione passo-passo della macchina di Von Neumann
  - Linguaggio Macchina (Una semplificazione)
  - Assembler (Una semplificazione)
- LINGUAGGIO MACCHINA (FORMATO BINARIO)
  - Esecuzione delle istruzioni e FASE di FETCH
  - Insieme di ISTRUZIONI (Qualche esempio)
  - Codifica dei DATI in memoria (Solo interi)
  - MODI di INDIRIZZAMENTO in memoria (Qualche esempio)
- LINGUAGGIO ASSEMBLER (ETICHETTE SIMBOLICHE)

LINGUAGGIO MACCHINA [1]

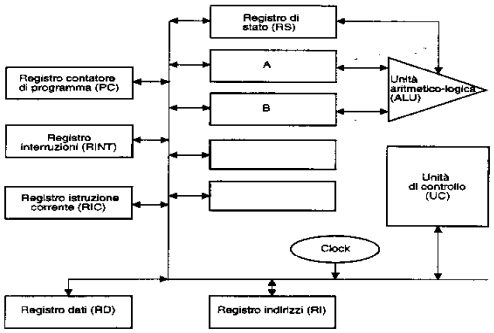
- ISTRUZIONI (PER NOI)
  - Codificata in una cella di memoria **[h bit]**
  - Codice operativo (Operazione da eseguire) **[m bit]**
  - Operando (Indirizzo dato su cui operare) **[n bit]**

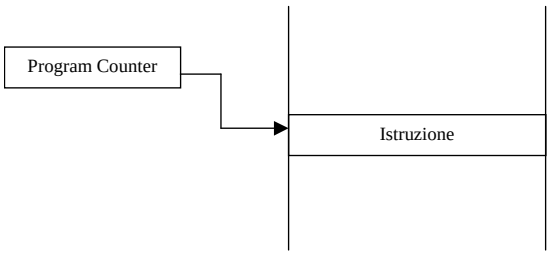


- CARATTERISTICHE LINGUAGGIO
  - Set di istruzioni (2^m possibilità)
  - Celle indirizzabili (2^n locazioni di memoria)
  - Possibile avere istruzioni su più celle di memoria ed utilizzare indirizzamenti indiretti

LINGUAGGIO MACCHINA [2]

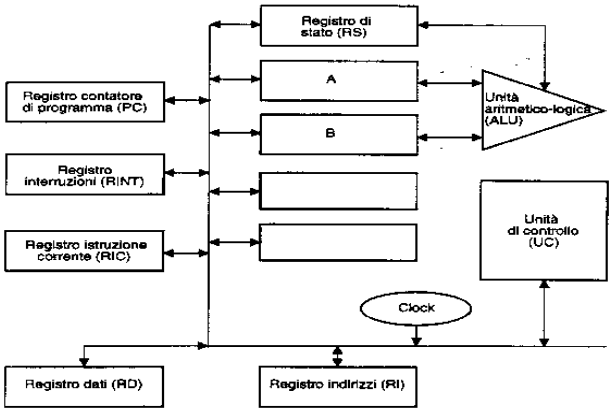
- ESECUZIONE (MICROISTRUZIONI)
  - Acquisizione dalla memoria **(fase di fetch)** dell'istruzione con caricamento della stessa in RIC ed aggiornamento PC
  - Interpretazione del codice operativo contenuto all'interno dell'istruzione
  - Esecuzione vera e propria dell'operazione





LINGUAGGIO MACCHINA [3]

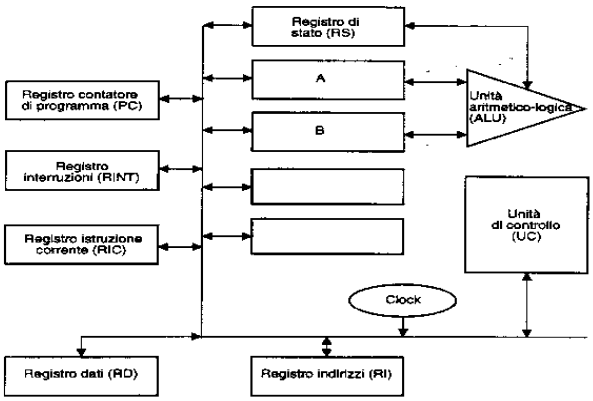
- FASE DI FETCH
  - PC -> RI
  - MEM[RI] -> RD
  - RD -> RIC
  - PC+1 -> PC
    (a parte jump)



- INTERPRETAZIONE
  - Elaboro da RIC il codice operativo e determino l'operazione da eseguire
  - Ogni operazione viene eseguita attraverso una serie di microistruzioni

LINGUAGGIO MACCHINA [4]

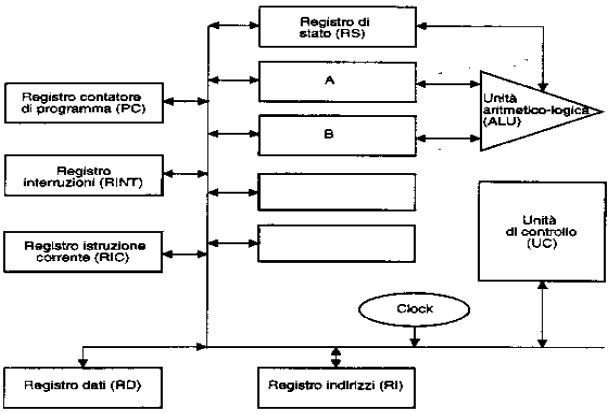
- CARICA NEI REGISTRI (LOADA *IND1*, LOADB *IND1*)
  - OP(RIC) -> RI
  - MEM[RI] -> RD
  - RD -> A
- SALVA IN MEMORIA (STOREA *IND1*, ...)
  - A -> RD
  - OP(RIC) -> RI
  - RD -> MEM[RI]



- LETTURA DA PERIFERICA (READ *IND1*)
  - RDP -> RD RDP (Registro Dati Periferica)
  - OP(RIC) -> RI
  - RD -> MEM[RI]

LINGUAGGIO MACCHINA [5]

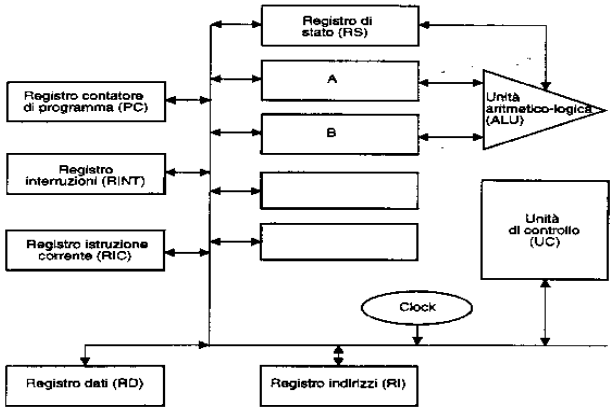
- SCRITTURA SU PERIFERICA (WRITE *IND1*)
  - OP(RIC) -> RI
  - MEM[RI] -> RD
  - RD -> RDP
- OPERAZIONI ALU (ADD, DIFF, MUL, DIV)
  - Risultato di A e B
  - Lo metto in A
  - Se ho resto, va in B



- ISTRUZIONE DI SALTO INCONDIZIONATO (JUMP *IND1*)
  - OP(RIC) -> PC

LINGUAGGIO MACCHINA [6]

- ISTRUZIONE DI SALTO CONDIZIONATO (JUMPZ *IND1*)
  - Se A=0 (RS bit zero)
  - OP(RIC) -> PC
- PAUSA (NOP)
  - Usata per far trascorrere un ciclo macchina a vuoto
- ARRESTO (HALT)
  - Termina il programma



LINGUAGGIO MACCHINA [7]

- ELENCO ISTRUZIONI E LORO POSSIBILE CODIFICA

0000	Loada
0001	Loadb
0010	Storea
0011	Storeb
0100	Read
0101	Write
0110	Add
0111	Dif
1000	Mul
1001	Div
1010	Jump
1011	Jumpz
1100	Nop
1101	Halt

- RAPPRESENTAZIONE DATI IN MEMORIA CENTRALE
  - Interi in complemento a 2 (in realtà molti tipi base)
  - Contenuti in celle di memoria con un loro indirizzo

LINGUAGGIO MACCHINA [8]

- PROGRAMMA
  - Istruzioni (dalla cella 0)
  - Dati (dopo le istruzioni)

- PROGRAMMA
  - 1 Programma per volta
  - Max 2^n tra istruzioni e dati
  - Max 2^n-*num(istr.)* in memoria
  - Solo programmi di tipo numerico
  - Poche istruzioni -> Molti spostamenti

LINGUAGGIO MACCHINA [9]

- ESEMPIO (CELLE A 16 BIT -> 4KB INDIRIZZABILI)
  - Moltiplicazione tra due numeri

Istruzioni m=4 n=12
Dati (-2^{15} $2^{15}-1$)

0	0100000000001000	0100 000000001000	Read 8
1	0100000000001001	0100 000000001001	Read 9
2	0000000000001000	0000 000000001000	Loada 8
3	0001000000001001	0001 000000001001	Loadb 9
4	1000000000000000	1000 000000000000	Mul
5	0010000000001000	0010 000000001000	Storea 8
6	0101000000001000	0101 000000001000	Write 8
7	1101000000000000	1101 000000000000	Halt
8	0000000000000000	0000000000000000	0
9	0000000000000000	0000000000000000	0

LINGUAGGIO MACCHINA [10]

- ESECUZIONE DI UN PROGRAMMA
  - PC = 0
- ISTR. 0: 0100|000000001000 (READ 8)
  - Fetch
  - Esecuzione: RDP -> RD
    OP(RIC) -> RI
    RD -> MEM[RI]
- ISTR. 1: 0100|000000001001 (READ 9)
  - Fetch
  - Esecuzione: RDP -> RD
    OP(RIC) -> RI
    RD -> MEM[RI]

LINGUAGGIO MACCHINA [11]

- ISTR. 2: 0000|000000001000 (LOADA 8)
  - Fetch
  - Esecuzione: OP(RIC) -> RI
    MEM[RI] -> RD
    RD -> A
- ISTR. 3: 0001|000000001001 (LOADB 9)
  - Fetch
  - Esecuzione: OP(RIC) -> RI
    MEM[RI] -> RD
    RD -> B
- ISTR. 4: 1000|000000000000 (MUL)
  - Fetch
  - Esecuzione: MUL

LINGUAGGIO MACCHINA [12]

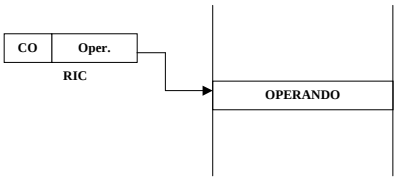
- ISTR. 5: 0010|000000001000 (STOREA 8)
  - Fetch
  - Esecuzione: A -> RD
    OP(RIC) -> RI
    RD -> MEM[RI]
- ISTR. 6: 0101|000000001000 (WRITE 8)
  - Fetch
  - Esecuzione: OP(RIC) -> RI
    MEM[RI] -> RD
    RD -> RDP
- ISTR. 7: 1101|000000000000 (HALT)
  - Fetch

LINGUAGGIO MACCHINA [13]

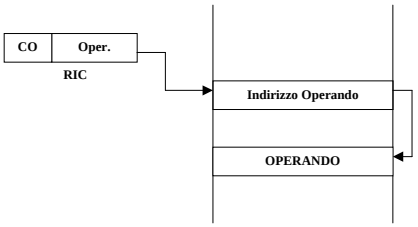
- LIMITAZIONI INDIRIZZAMENTO DIRETTO
  - Problemi con spostamento in memoria dei programmi (rilocazione)
  - Con memoria grande ho problemi ad indirizzare le celle contenenti i dati (limitato a 2^n)
- MODI DI INDIRIZZAMENTO
  - Indirizzamento DIRETTO
  - Indirizzamento INDIRETTO
  - Indirizzamento tramite REGISTRO INDICE
  - Indirizzamento IMMEDIATO

LINGUAGGIO MACCHINA [14]

- INDIRIZZAMENTO DIRETTO
  - Il campo operando contiene l'indirizzo (assoluto) della cella di memoria dove è contenuto il dato
  - Limite a 2^n

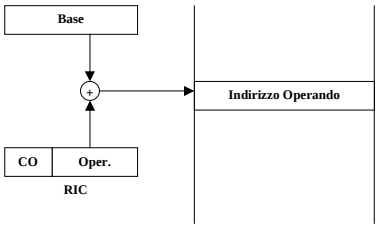


- INDIRIZZAMENTO INDIRETTO
  - Nella cella puntata dal campo operando trovo l'indirizzo (assoluto) in memoria del dato
  - Indirizzo 2^h celle di memoria
  - Necessita di un accesso in più alla memoria



LINGUAGGIO MACCHINA [15]

- INDIRIZZAMENTO TRAMITE INDIRIZZO INDICE
  - Sommo il campo operando al contenuto di un registro ed accedo così all'operando
  - Uso un solo accesso
  - Indirizzo 2^h celle di memoria
  - Non ho problemi per rilocare



- INDIRIZZAMENTO IMMEDIATO
  - Il campo operando contiene già l'operando che devo utilizzare
  - Non richiede accessi in memoria
  - Limitato numero di operandi

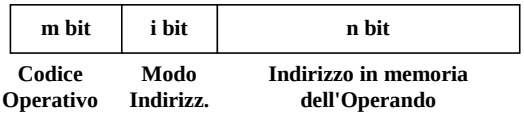
LINGUAGGIO MACCHINA [16]

- CODIFICA
  - Aumento il numero delle istruzioni in modo che l'Unità di controllo capisca come accedere all'operando, complica inutilmente il set di istruzioni [Non Usato]
  - Inserisco nella codifica dell'istruzione un campo che codifichi il modo di indirizzamento

00	Diretto
01	Indiretto
10	Registro Indice
11	Immediato

h = m + i + n

Istruzione [h bit]



LINGUAGGIO ASSEMBLER [1]

- ASSEMBLER
  - Programmo ancora a Basso Livello
  - Corrispondenza biunivoca tra Assembler e Linguaggio Macchina
  - Traduco in binario tramite assemblatore
  - Ricerca di una maggior efficienza
  - Uso parole chiave ed etichette, disinteressandomi della loro codifica binaria
  - Astrazione dalla codifica binaria
  - Etichette simboliche per riferirsi agli indirizzi di istruzioni e dati
  - Possibilità, grazie alle etichette, di posizionare il codice dove voglio per la rilocazione

LINGUAGGIO ASSEMBLER [2]

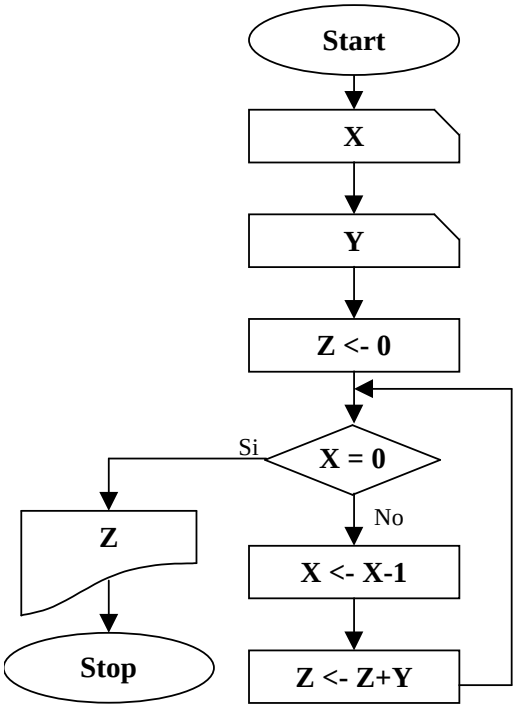
- ISTRUZIONI
  - Uso le parole chiave introdotte in anticipo
- MODI DI INDIRIZZAMENTO (NUM = ETICHETTA)
  - Diretto: LOADA NUM
  - Indiretto: LOADA @NUM
  - Registro Indice: LOADA NUM(I)
  - Immediato: LOADA #NUM
- TRADUZIONE (POSTO NUM = 9)
  - LOADA NUM: 0000 00|00 0000 1001
  - LOADA @NUM: 0000 01|00 0000 1001
  - LOADA NUM(I): 0000 10|00 0000 1001
  - LOADA #NUM: 0000 11|00 0000 1001

LINGUAGGIO ASSEMBLER [3]

- ESEMPIO 1: MOLTIPLICAZIONE PER SOMME RIPETUTE
  - **Specifica Informale**: supposto di non avere l'operatore di moltiplicazione si vuole implementare tale operazione attraverso la ripetizione di somme.
  - **Specifica Semi Formale**: uso tre variabili (X, Y, Z), al termine del programma Z conterrà X•Y; per ottenere il risultato incremento Z del valore Y per X volte decrementando X ad ogni ciclo
  - **Specifica Formale**: uso diagrammi di Jackson per specificare il flusso di controllo e di elaborazione dei dati
  - **Codifica**: uso linguaggio assembler
  - **Traduzione**: traduco il codice assembler in linguaggio macchina

LINGUAGGIO ASSEMBLER [4]

- DIAGRAMMI DI JACKSON



LINGUAGGIO ASSEMBLER [5]

- CODICE ASSEMBLER:

```
        READ    X
        READ    Y
        LOADA   ZERO
        STOREA  Z
TEST    LOADA   X
        JUMPZ   FINE
        LOADB   UNO
        DIF
        STOREA  X
        LOADA   Y
        LOADB   Z
        ADD
        STOREA  Z
        JUMP    TEST
FINE    WRITE   Z
        HALT
ZERO    0
UNO     1
X       INT
Y       INT
Z       INT
```

LINGUAGGIO ASSEMBLER [6]

- CODICE MACCHINA:

```
0     0100  00 00  0001  0010
1     0100  00 00  0001  0011
2     0000  00 00  0001  0000
3     0010  00 00  0001  0100
4     0000  00 00  0001  0010
5     1011  00 00  0000  1100
6     0001  00 00  0001  0001
7     0111  00 00  0000  0000
8     0010  00 00  0001  0010
9     0000  00 00  0001  0011
10    0001  00 00  0001  0100
11    0110  00 00  0000  0000
12    0010  00 00  0001  0100
13    1010  00 00  0000  0100
14    0101  00 00  0001  0100
15    1101  00 00  0000  0000
16    0000  0000  0000  0000
17    0000  0000  0000  0001
18    0000  0000  0000  0000
19    0000  0000  0000  0000
20    0000  0000  0000  0000
```

LINGUAGGIO ASSEMBLER [7]

- ESEMPIO 2: RADICE QUADRATA INTERA
  - **Specifica Informale**: si vuole calcolare la radice quadrata di un numero intero dato avendo a disposizione gli operatori MUL e JUMPGEZ (1110). Inizializzo con indirizzamento immediato.
- ESEMPIO 3: INVERSIONE DI UNA LISTA DI NUMERI
  - **Specifica Informale**: si vuole leggere una lista di numeri (terminata dallo zero) e riproporla a video esttamente nell'ordine inverso.
- ESEMPIO 4: MEDIA ARITMETICA DI UNA SERIE DI NUMERI
  - **Specifica Informale**: si legge una lista di numeri (terminata dallo 0) e se ne calcola la media aritmetica e la si visualizza.

DIAGRAMMI DI JACKSON [1]

- ALGORITMO
  - Insieme di azioni semplici (o minime) per la risoluzione di un problema
  - Decomposizione semplice di un problema complesso in problemi più semplici (di soluzione nota)
- DIAGRAMMI DI JACKSON
  - Rappresentazione del flusso di controllo di un programma e delle operazioni logico/fisiche sui dati