Università degli Studi di Cagliari
Corsi di Laurea in Ingegneria Chimica e Ingegneria Meccanica

FONDAMENTI DI INFORMATICA

http://people.unica.it/gianlucamarcialis

A.A. 2018/2019

Docente: **Gian Luca Marcialis**

ESERCITAZIONE PYTHON
ARCHITETTURA DEI CALCOLATORI

Sommario

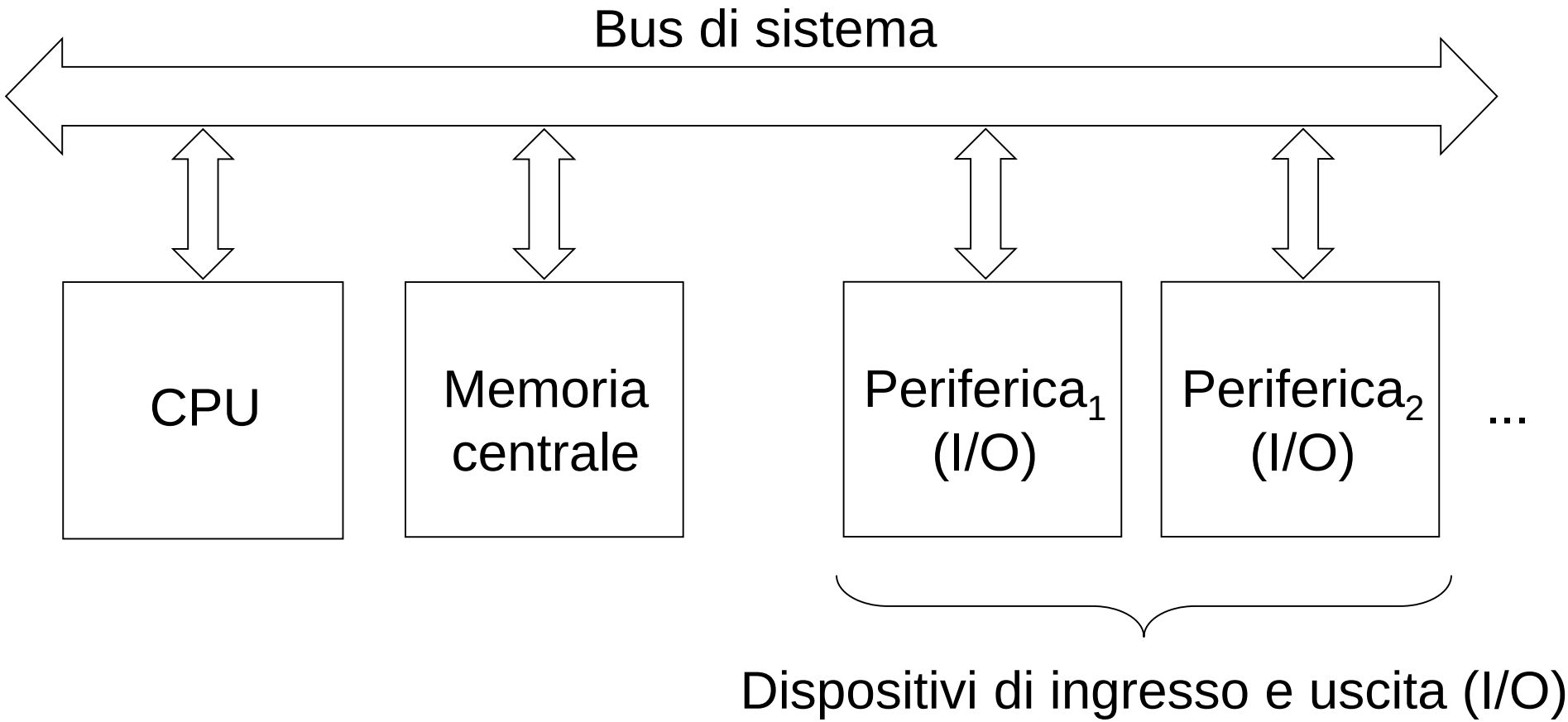
- Funzionamento del calcolatore: ripasso
- Un calcolatore giocattolo
- Implementazione in Python

Algoritmi e programmi

- Ogni calcolatore mette a disposizione un certo numero (finito) di operazioni elementari su dati rappresentati in codifica binaria
- Qualsiasi algoritmo deve essere espresso come sequenza di operazioni elementari effettivamente eseguibili dal calcolatore (**programma**)

  es.: molti calcolatori non forniscono l'operazione di estrazione della radice quadrata, ma solo le operazioni di somma e prodotto; l'estrazione della radice quadrata deve essere espressa come sequenza di somme e prodotti

Architettura di Von Neumann



Schema di funzionamento della macchina di Von Neumann

- I programmi sono composti da istruzioni codificate in binario:
  - istruzioni di elaborazione (ad es. operazioni numeriche)
  - istruzioni di trasferimento di dati tra due componenti della macchina
- Il funzionamento della macchina di Von Neumann è un ciclo continuo:
  - la CPU estrae le istruzioni e i dati dalla memoria principale...
  - ...le decodifica (determina l'operazione da eseguire e i gli operandi)...
  - ...e le esegue

Esecuzione di un programma: funzionamento elementare

- Si carica in memoria centrale il programma in codice *binario*
  - il programma occupa una sequenza **contigua** di "word" di memoria
- Ogni istruzione è sottoposta, sequenzialmente, a un "ciclo di esecuzione"
- L'ultima istruzione indica esplicitamente il termine delle operazioni ("halt")

Registri della CPU coinvolti

- PC = Program Counter
  - Contiene, all'istante t, l'indirizzo dell'istruzione da eseguire all'istante successivo
- IR = Instruction Register
  - Contiene, all'istante t, l'istruzione in esecuzione
- ACC = Service Register o Accumulator
  - Contiene dati parziali o finali trasferiti/da trasferire dalla/in memoria
  - Possono essere più d'uno

Set di istruzioni

- Le istruzioni sono memorizzate in un'area permanente (ROM), e sono indirizzate attraverso un «codice operativo», una stringa di bit che rappresenta quel che l'istruzione deve fare
- Noi useremo codici operativi «simbolici»

Ciclo di esecuzione delle istruzioni

- **Istruction Fetch (IF): Prelievo ("fetch") delle istruzioni**
  - La CPU legge un'istruzione dalla memoria
  - M[PC] ➔ IR
  - PC ← PC + 1
- **Istruction Decode (ID): Interpretazione delle istruzioni**
  - L'istruzione viene decodificata per determinare quale azione è stata richiesta
  - Il sistema "separa" l'azione dagli operandi
- **Operand Fetch (OD): Prelievo ("fetch") dei dati**
  - L'esecuzione dell'istruzione potrebbe richiedere dei dati dalla memoria o da un dispositivo di I/O
- **Execute (EX): Elaborazione dei dati**
  - Può essere richiesta l'esecuzione di operazioni aritmetiche o logiche sui dati.
  - I valori degli operandi sono trasferiti in registri locali della ALU
- **Operand Store (OS): Scrittura dei dati**
  - Si può richiedere di trasferire i dati elaborati in memoria o ad un modulo di I/O.
  - Registri locali ALU ➔ MBR ➔ Memoria
- **Passaggio all'istruzione successiva**

Insieme di istruzioni di una CPU

- Istruzioni di **calcolo**
  - operazioni aritmetiche (somma, sottrazione, prodotto, divisione ecc.)
  - operazioni logiche (algebra booleana)
  - L'elaborazione dei dati avviene nella ALU
  - Una ALU è in grado di eseguire un insieme di operazioni predefinito (in fase di progetto) su dati codificati in binario
- Istruzioni di **prelievo** dati dalla memoria o dalle periferiche
- Istruzioni di **trasferimento** dati su memoria o su periferiche

- I **programmi** eseguibili dal calcolatore sono espressi come *sequenza di istruzioni codificate in binario*, ognuna corrispondente ad una delle operazioni precedenti
  - linguaggio macchina

Esecuzione di un programma: funzionamento elementare

- Si carica in memoria centrale il programma in codice *binario*
  - il programma occupa una sequenza **contigua** di "parole" di memoria
- Ogni istruzione è sottoposta, sequenzialmente, a un "ciclo di esecuzione"
- L'ultima istruzione indica esplicitamente il termine delle operazioni ("halt")

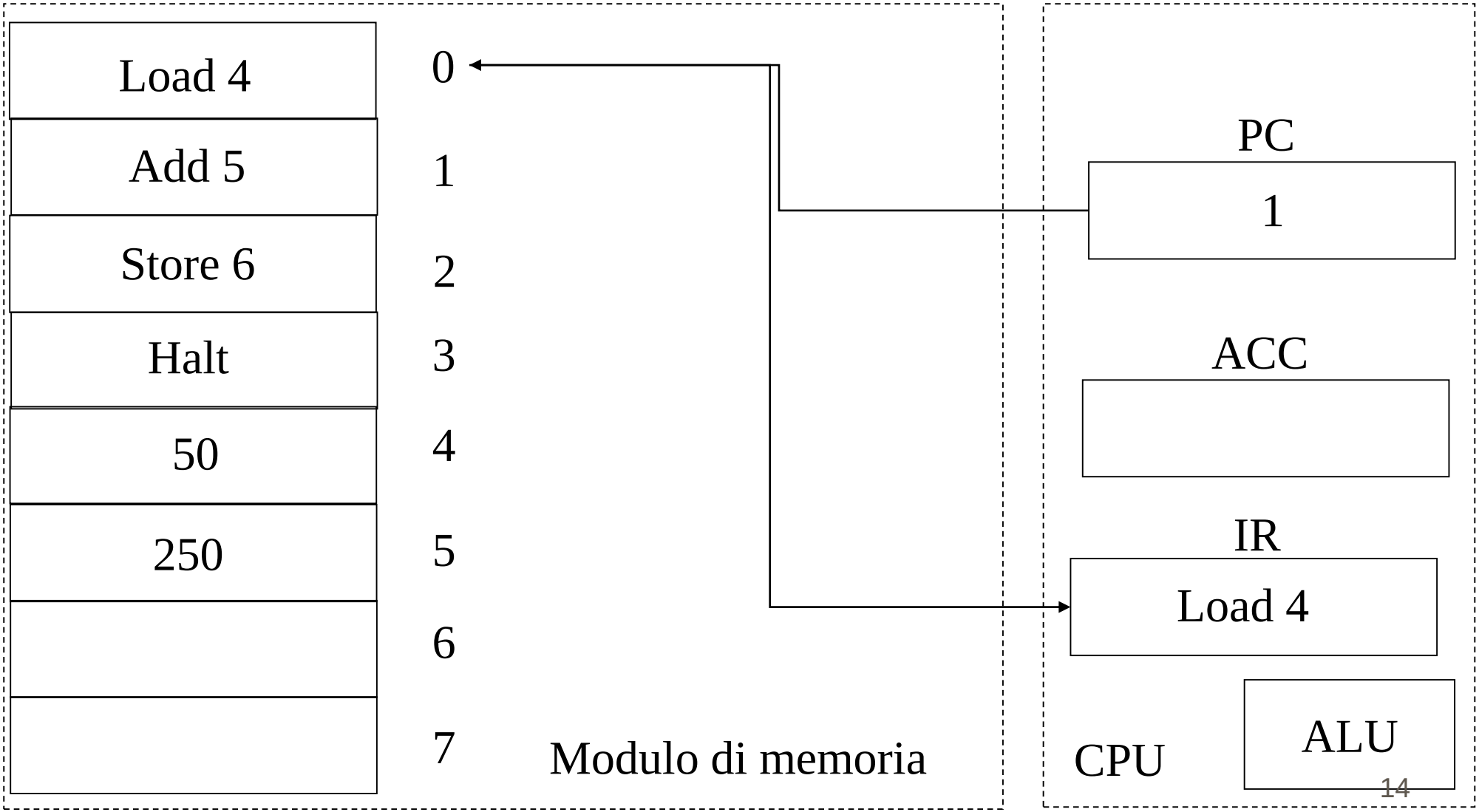
Un semplice esempio in “linguaggio macchina”

- Disponiamo di un modulo di memoria a 8 indirizzi
- Supponiamo che la nostra architettura disponga delle seguenti istruzioni
  - Load X ➔ Carica in un registro di servizio (ACC) il valore all’indirizzo X
  - Add Y ➔ Somma al contenuto di ACC il valore all’indirizzo Y
  - Store X ➔ Memorizza il contenuto di ACC nell’indirizzo X
  - Halt ➔ il ciclo di esecuzione termina

Un semplice esempio in linguaggio macchina

- Scriviamo un algoritmo per la somma di due numeri A e B che sono memorizzati rispettivamente in 4 e 5, e vogliamo che la somma sia memorizzata in 6
- Una possibile soluzione è:
  - Load 4 (carico in ACC il valore di A)
  - Add 5 (gli sommo il valore di B)
  - Store 6 (memorizzo il risultato in 110)
  - Halt (l'algoritmo termina)
- Il programma viene caricato nella memoria centrale a partire dalla prima locazione disponibile
- Parte quindi il ciclo di esecuzione delle istruzioni

Ciclo di esecuzione delle istruzioni: prelievo della prima istruzione

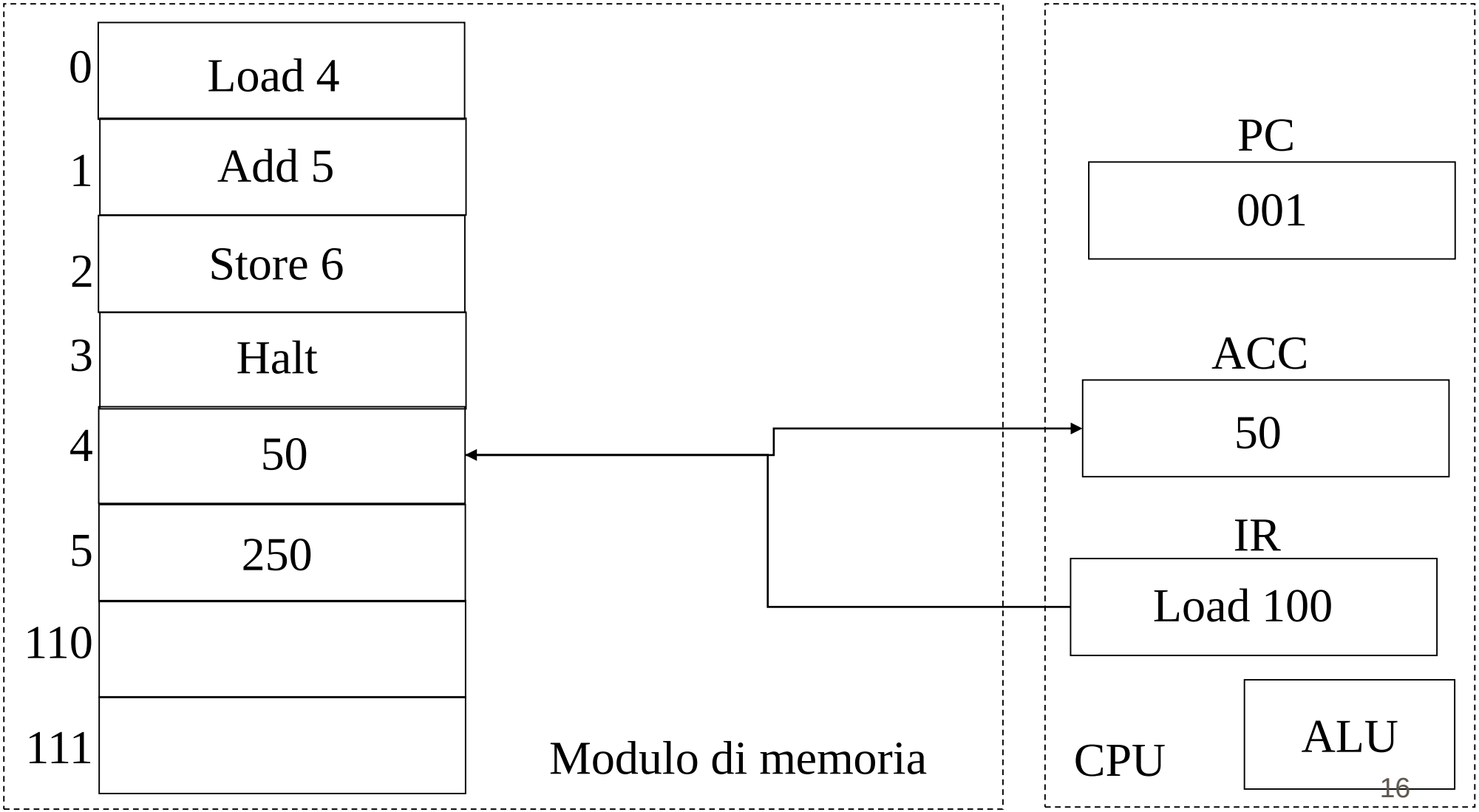


Ciclo di esecuzione delle istruzioni: intepretazione della prima istruzione

Codice operativo (opcode): viene identificata una sequenza di segnali appropriata per attivare il trasferimento dati da memoria a registro

E' l'indirizzo dell'operando, verrà prelevato nella fase successiva

Ciclo di esecuzione delle istruzioni: prelievo dell'operando, elaborazione e risultato



Un simulatore in Python

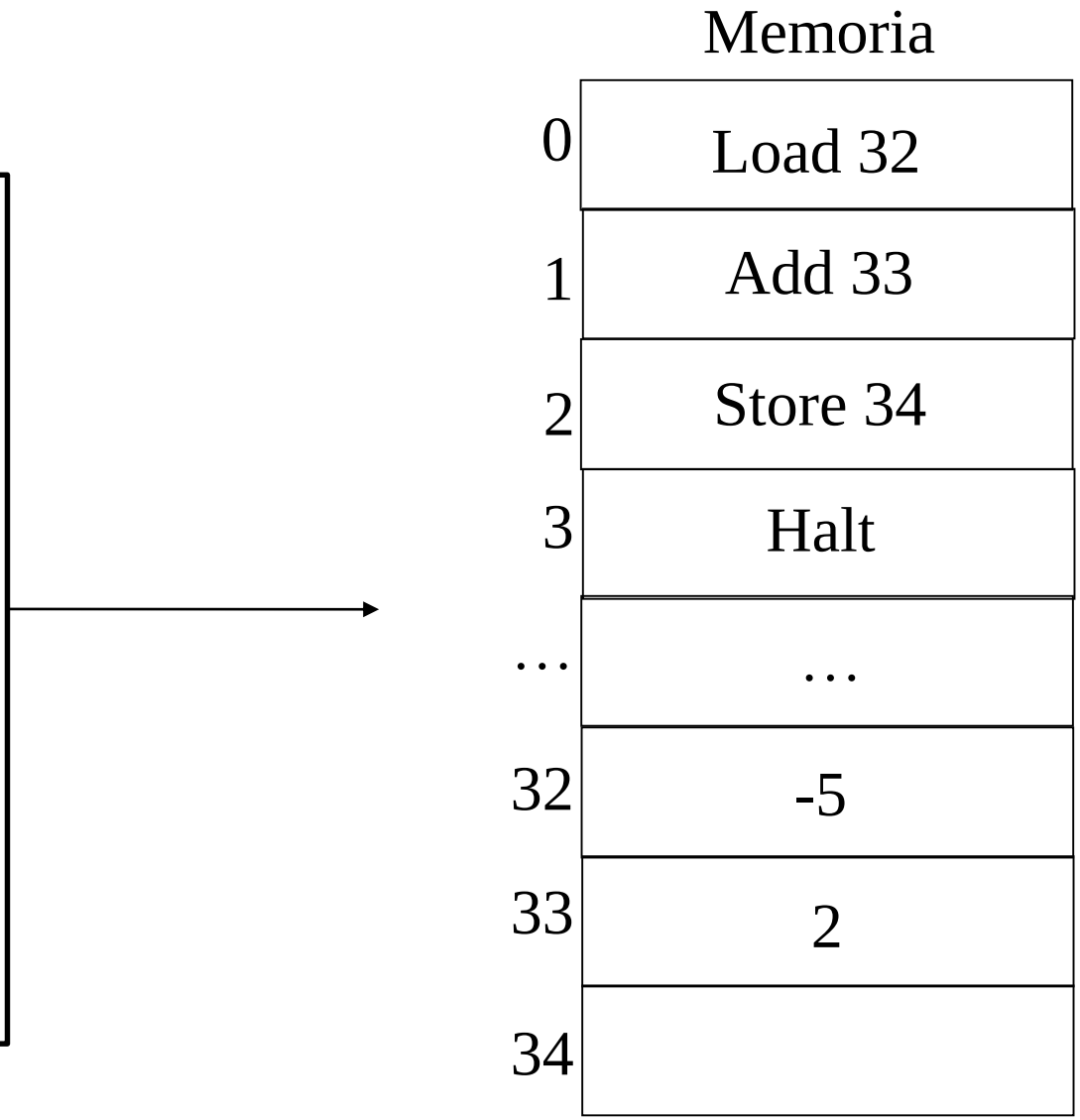
- Implementare in Python un simulatore di calcolatore elettronico composto da PC, IR e ACC
- Una memoria a 64 parole per le istruzioni e i dati
- Funzionalità:
  - Legge un algoritmo nella forma di set di istruzioni da file
  - Legge i dati, separati dalla parola-chiave «dati»
  - Memorizza le istruzioni dall'indirizzo 0 all'indirizzo 31
  - Memorizza i dati dall'indirizzo 32 al 63
- Dopo di che avvia l'esecuzione secondo la procedura vista nell'esempio

Istruzioni del calcolatore

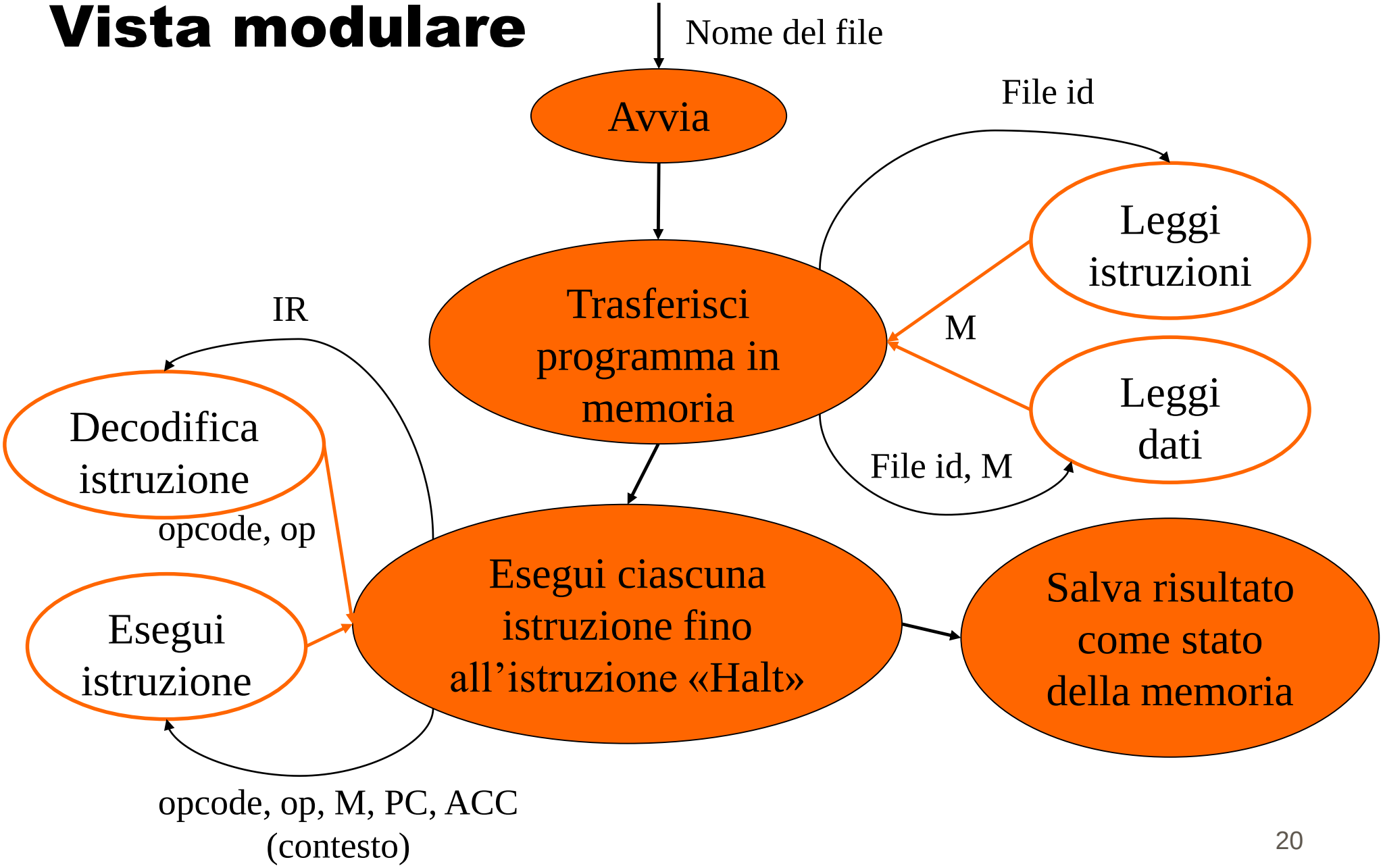
- Per iniziare, doteremo il calcolatore delle istruzioni-base:
  - LOAD X
  - STORE X
  - ADD X
  - HALT
- Volendo incrementare la sua capacità di calcolo, ne aggiungeremo altre all'occorrenza, sempre sfruttando i registri a disposizione:
  - Aggiungendo istruzioni di condizione e salto per implementare if-else e while

Esempio di file: «PCsum.txt»

```
LOAD 32
ADD 33
STORE 34
HALT
dati
-5
2
```



Vista modulare



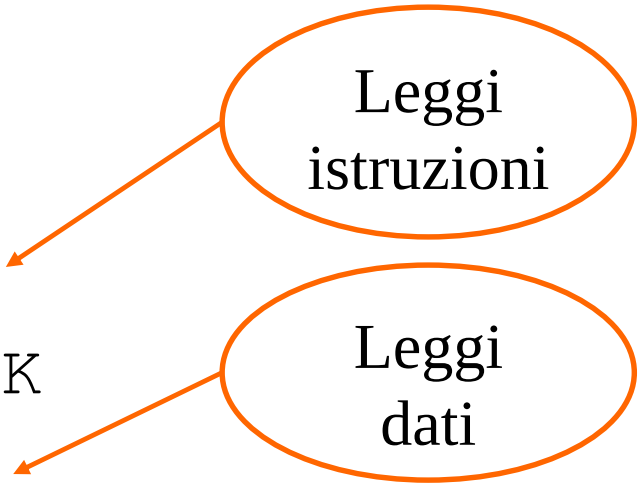
Progetto del calcolatore

- Ci serve prima di tutto una funzione di avvio, che ci permetta di «lanciare» il programma letto da file:

```
def avvia(nomefile):
        PC=0  #contatore di programma
        ACC=0 #accumulatore o registro di servizio
        IR=0  #registro istruzione

        M=trasferisci(nomefile) #M è la memoria
        if M: #se la memoria non è 'vuota'...
                IR=M[PC] #preleva istruzione e depositala nel IR
                while IR!="HALT":
                        PC=PC+1
                        opcode, op=decodifica(IR)
                        M,PC,ACC=esegui(opcode,op,M,PC,ACC) #esecuzione
                        IR=M[PC] #prelievo dell'istruzione successiva
                salvaMemoria(M)
```

Trasferimento dei dati

```
def trasferisci(nomefile):
    idf=open(nomefile,"r")

    M=leggi_istruzioni(idf)
    if M: #se la lettura è OK
        M=leggi_dati(idf,M)

    idf.close()
    return M
```



Lettura delle istruzioni

```
def leggi_istruzioni(idf):
      M=["Empty"] * 64 #memoria di 64 parole

      linea=idf.readline()
      if linea=="":
            return False
      i=0
      linea=linea[0:len(linea)-1]
      while linea!="dati":
            M[i]=linea
            i=i+1
            linea=idf.readline()
            linea=linea[0:len(linea)-1]
      return M
```

Lettura dei dati

```
def leggi_dati(idf,M):
    i=32 #memorizzati dall'indirizzo 32

    linea=idf.readline()
    while linea!="":
        M[i]=int(linea) #interi
        i=i+1
        linea=idf.readline()

    return M
```

Progetto del calcolatore

- Ci serve prima di tutto una funzione di avvio, che ci permetta di «lanciare» il programma letto da file:

```
def avvia(nomefile):
        PC=0  #contatore di programma
        ACC=0 #accumulatore o registro di servizio
        IR=0  #registro istruzione

        M=trasferisci(nomefile) #M è la memoria
        if M: #se la memoria non è 'vuota'...
                IR=M[PC] #preleva istruzione e depositala nel IR
                while IR!="HALT":
                        PC=PC+1
                        opcode, op=decodifica(IR)
                        M,PC,ACC=esegui(opcode,op,M,PC,ACC) #esecuzione
                        IR=M[PC] #prelievo dell'istruzione successiva
                salvaMemoria(M)
```

Decodifica dell'istruzione

```
def decodifica(istruzione):
    istruzione=istruzione.split()

    opcode=istruzione[0] #codice operativo
    if len(istruzione)>1:
        op=int(istruzione[1]) #operando
    else:
        op=""

    return opcode,op
```

Esecuzione dell'istruzione implementazione della memoria «ROM»

```
def esegui(opcode,op,M,PC,ACC):
    if opcode=="LOAD":
        ACC=M[op]
    elif opcode=="STORE":
        M[op]=ACC
    elif opcode=="ADD":
        ACC=ACC+M[op]

    #restituisce il nuovo "stato"
    return M,PC,ACC
```

Progetto del calcolatore

- Ci serve prima di tutto una funzione di avvio, che ci permetta di «lanciare» il programma letto da file:

```
def avvia(nomefile):
        PC=0  #contatore di programma
        ACC=0 #accumulatore o registro di servizio
        IR=0  #registro istruzione

        M=trasferisci(nomefile) #M è la memoria
        if M: #se la memoria non è 'vuota'...
                IR=M[PC] #preleva istruzione e depositala nel IR
                while IR!="HALT":
                        PC=PC+1
                        opcode, op=decodifica(IR)
                        M,PC,ACC=esegui(opcode,op,M,PC,ACC) #esecuzione
                        IR=M[PC] #prelievo dell'istruzione successiva
                salvaMemoria(M)
```

Salvataggio dello stato del calcolatore

```
def salvaMemoria(M):
    address=0
    idf=open("PCoutput.txt","w")

    for parola in M:
        idf.write("%2d  %s\n"  %  (address,
str(parola)))
        address=address+1

    idf.close()
```

Avvio!

- Avviamo il programma dall'editor (RUN)
- Invochiamolo usando la funzione avvia()

```
>>> avvia("PCsum.txt")
```

Per arricchire il set istruzioni aritmetiche

- Potremmo pensare ad una funzione ALU che appunto simula l'operato dell'unità logico-aritmetica del calcolatore in base all'operando ed all'opcode:

```
def ALU(opcode, op, M, ACC):
    if opcode=="ADD"
        ACC=ACC+M[op]
    elif opcode== "SUB"
        ACC=ACC-M[op]
    elif opcode== "EQ"
        ACC=ACC==M[op] #cosa fa?
    return ACC
```

Modifica di esegui

```
def esegui(opcode,op,M,PC,ACC):
    if opcode=="LOAD":
        ACC=M[op]
    elif opcode=="STORE":
        M[op]=ACC
    elif opcode=="...":
    #altre istruzioni non aritmetiche
    else:
        ACC=ALU(opcode,op,M,ACC)
  #restituisce il nuovo "stato"
  return M,PC,ACC
```

Aggiunta di potere espressivo al nostro calcolatore

- Volendo fare esercizi più complessi, tipo implementazione del «massimo», possiamo dotare la memoria ROM (funzione `esegui()`) di più istruzioni

- **Esercizio.** Aggiungere le seguenti nella funzione esegui():

SUB X ➔ sottrae all'ACC il valore M[X]

INF ➔ pone il valore di ACC a 1 se ACC<0

BEQ X ➔ salta all'istruzione di indirizzo X se ACC==0

JUMP X ➔ salta all'istruzione di indirizzo X

- Nota bene: «saltare» significa alterare forzatamente il contenuto del PC, ovvero assegnarlo al valore X anziché al suo valore naturale (prossima istruzione da eseguire). Per esempio, implementare **`JUMP X`** significa imporre semplicemente:

`PC = X`

L'algoritmo del massimo in linguaggio macchina

Adesso siamo pronti a implementare il famoso algoritmo del massimo:

```
d=x-y
if d<0:
    max=y
else:
    max=x
```

```
0 LOAD 32
1 SUB 33
2 INF
3 BEQ 7
4 LOAD 33
5 STORE 34
6 JUMP 9
7 LOAD 32
8 STORE 34
9 HALT
```

Esempio di file per il massimo

```
LOAD 32
SUB 33
INF
BEQ 7
LOAD 33
STORE 34
JUMP 9
LOAD 32
STORE 34
HALT
dati
5
2
```