Corso di Calcolatori Elettronici A

Introduzione

2007/2008

Sito Web:http://prometeo.ing.unibs.it/quarella

Prof. G. Quarella

prof@quarella.net

Il calcolatore digitale

- Il calcolatore digitale è una macchina che risolve problemi a livello simbolico eseguendo un insieme di istruzioni fornitegli da un essere umano
- L'insieme di istruzioni si chiama programma

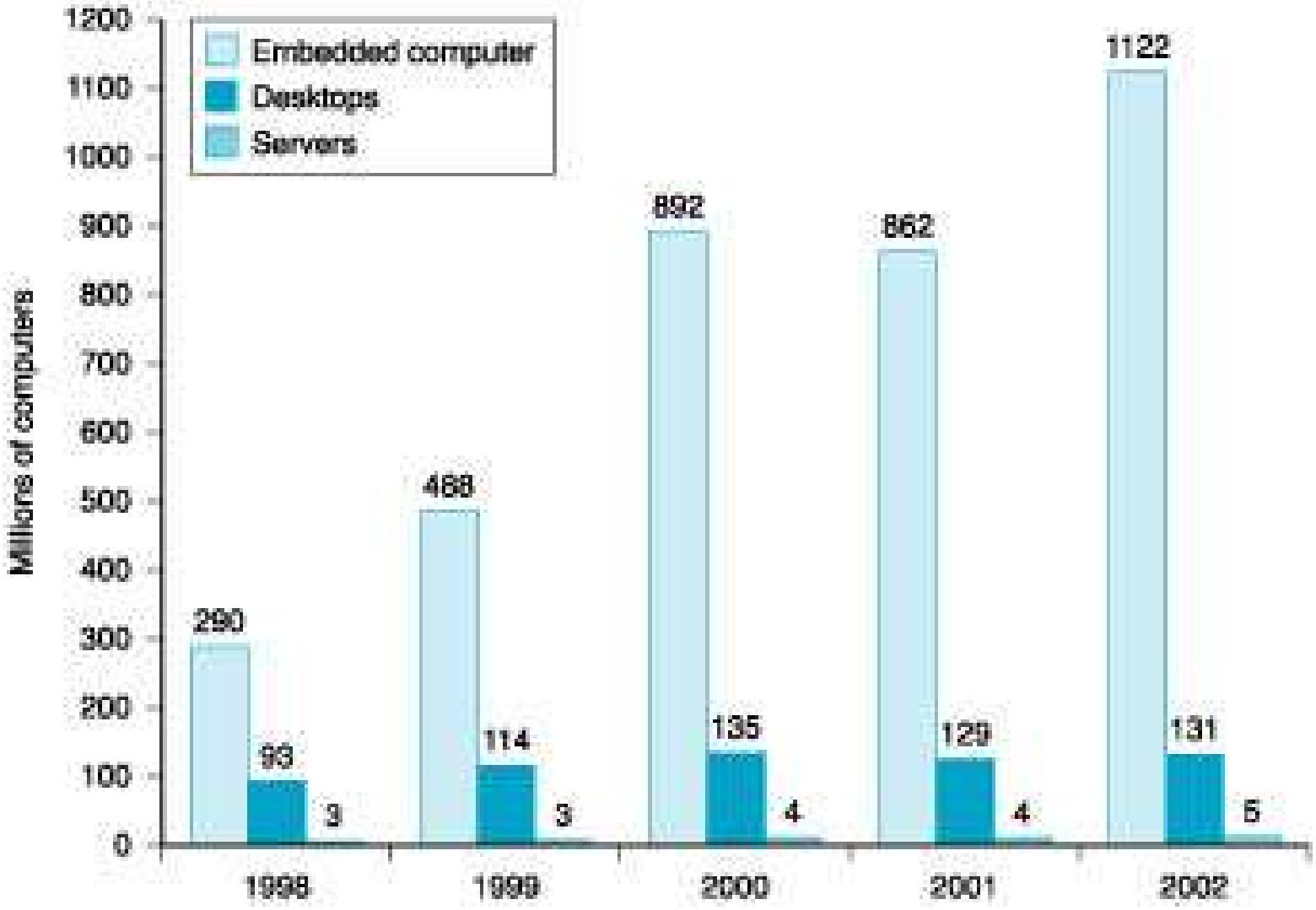
Il calcolatore elettronico

- Un calcolatore digitale può essere realizzato usando diverse tecnologie: noi ci occupiamo di calcolatori elettronici ovvero calcolatori realizzati mediante circuiti elettronici.
- I segnali elettrici vengono associati a numeri binari
- Es.: **transistor** on – off, 0 - 1

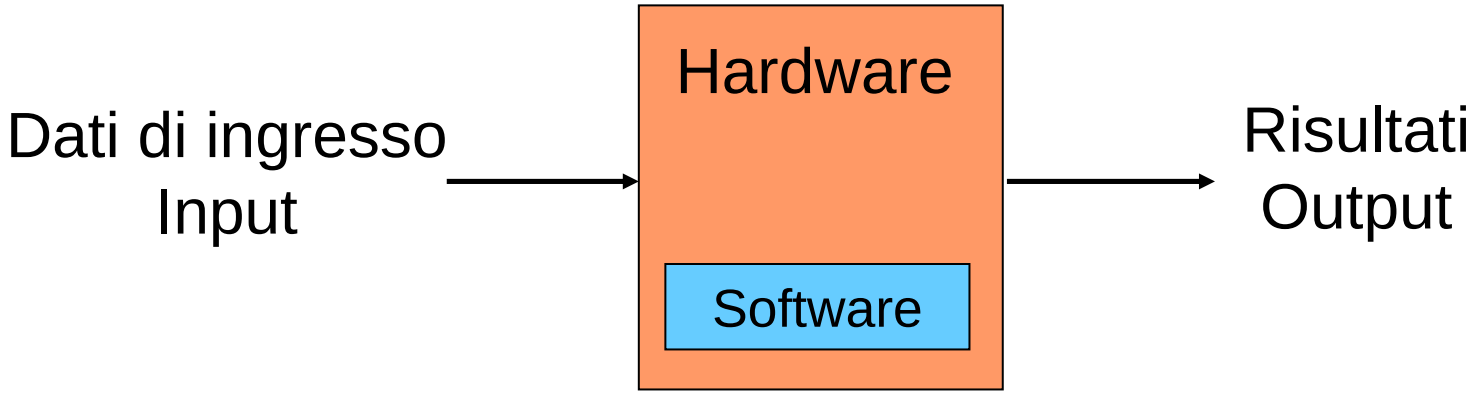
Classi di computer

- Desktop computer
- Server
  - Web server, Database server, ecc.
  - Supercomputer
- Embedded computer
  - Cellulari, elettrodomestici, ecc.

Processori venduti tra il 1998 e il 2002



Il calcolatore elettronico

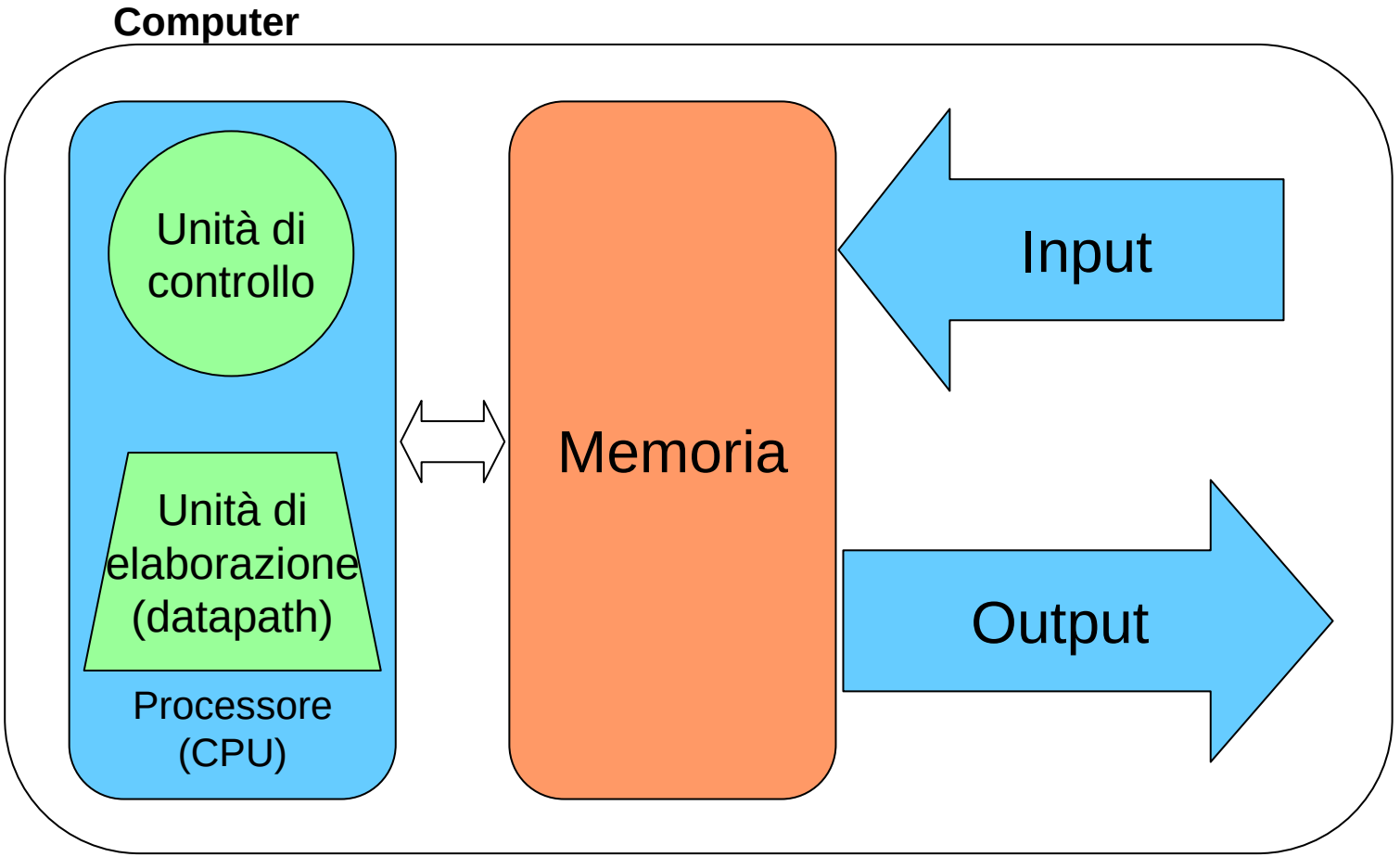


Il programma è memorizzato e definisce lo stato del calcolatore.

Il programma viene 'interpretato' dall'hardware

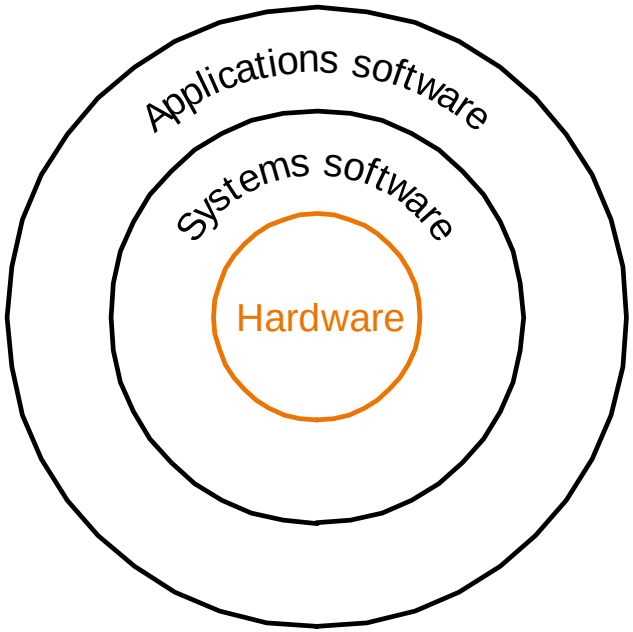
Hardware

Componenti di un calcolatore



Software

Organizzazione gerarchica



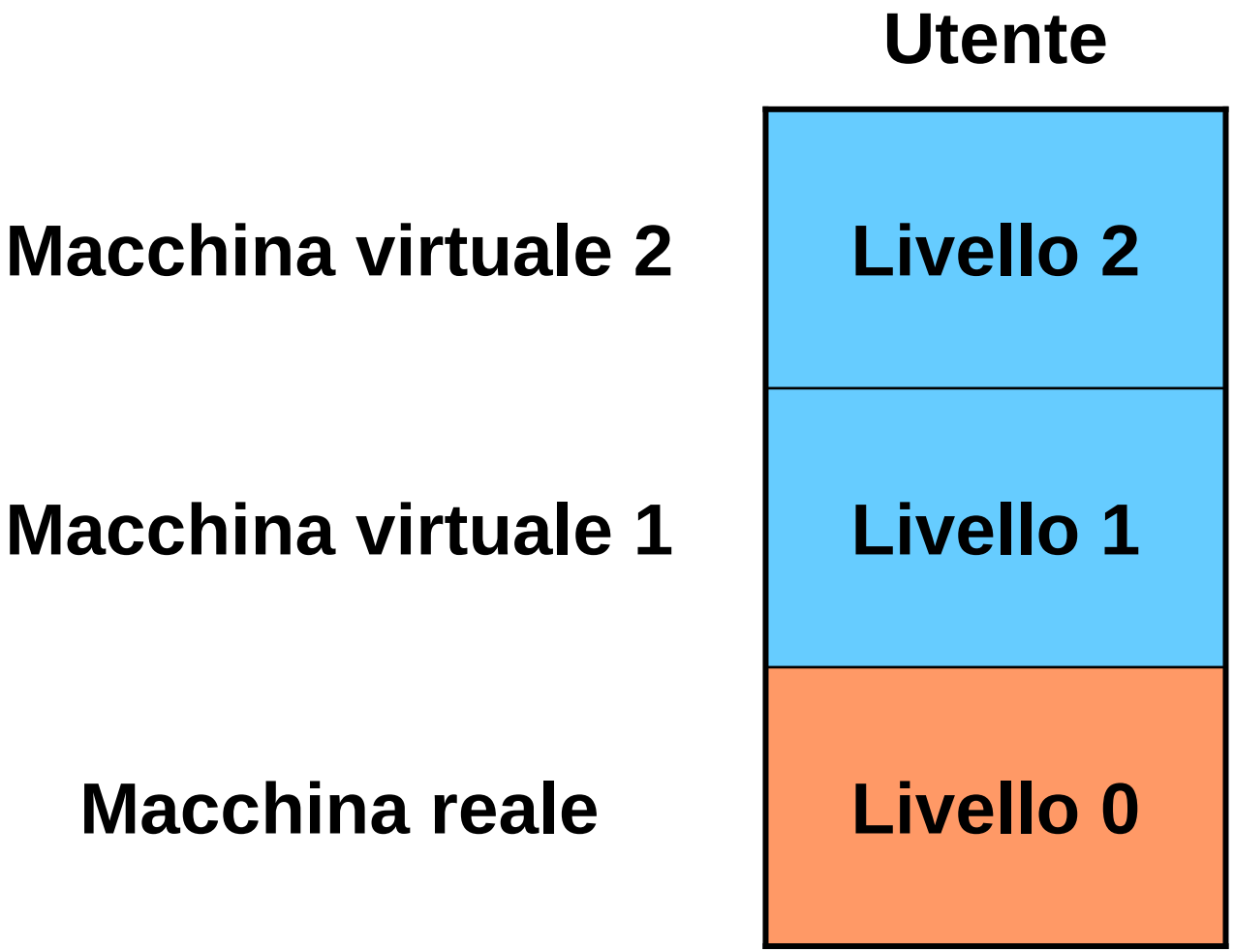
Software di sistema:

- sistema operativo
- compilatori

Livelli di astrazione

- I livelli di astrazione o astrazioni sono un principio di progetto comune nella progettazione di sistemi hardware o software
- Per dominare la complessità di un sistema lo si scompone in livelli gerarchici:
  - Ogni livello inferiore nasconde i dettagli (l'implementazione) del sottosistema al livello superiore (information hiding - incapsulamento)
  - L'**interfaccia** tra due livelli è ciò che il livello superiore deve conoscere del livello inferiore adiacente

Livelli di astrazione



Livelli di astrazione

- L'**applicazione** nasconde all'utente tutti i dettagli di basso livello
- Il **sistema operativo** nasconde alle applicazioni (e a chi le ha progettate) i dettagli relativi alle operazioni di I/O, all'allocazione della memoria, ecc.
- L'**hardware** nasconde la sua implementazione presentando un'interfaccia con tutto quello che un programmatore deve sapere per scrivere un programma in linguaggio macchina

L’interfaccia tra hardware e software

ISA: Instruction Set Architecture

L’architettura del set di istruzioni è l’interfaccia fra l’hardware ed il software di più basso livello.

Le componenti di progetto dell’hardware che devono essere comprese per scrivere in linguaggio macchina, ovvero le istruzioni, i registri, i dispositivi di I/O, ecc.

Architetture attuali: IA-32, PowerPC, MIPS, SPARC, ARM, ecc.

I processori possono presentare la medesima ISA pur avendo implementazioni differenti:

- **IA-32: Intel, AMD**

Linguaggio Macchina

- I circuiti elettronici di un calcolatore sono in grado di riconoscere ed eseguire un numero limitato di tipi di istruzioni
- Le istruzioni sono costituite da parole di bit ovvero numeri binari
- L’insieme di queste istruzioni costituisce il linguaggio macchina
- Molto distante dal linguaggio umano
- Serve una rappresentazione simbolica per rendere il codice comprensibile al programmatore

Linguaggio Assembler

Linguaggio assemblatore / Assembly language

00100010001100010000000000000100 ↔ addi $s1, $s1, 4

- Insieme di codici mnemonici associati ai codici operativi e i registri
- L'assemblatore traduce il linguaggio assembler in linguaggio macchina
- Aiuti per il programmatore:
  - **Etichette per indirizzi di memoria**
  - **Pseudoistruzioni**
  - **Macro**
  - **Direttive**
  - **Commenti**

Programmazione in linguaggio assembler

- Nessuna portabilità
- Molte linee di codice: bassa produttività
- Scelta obbligata se non sono disponibili linguaggi di alto livello per una determinata ISA
- Talvolta utile per ottimizzare piccole porzioni di codice (prodotte da un compilatore) di applicazioni critiche (vincoli real-time)

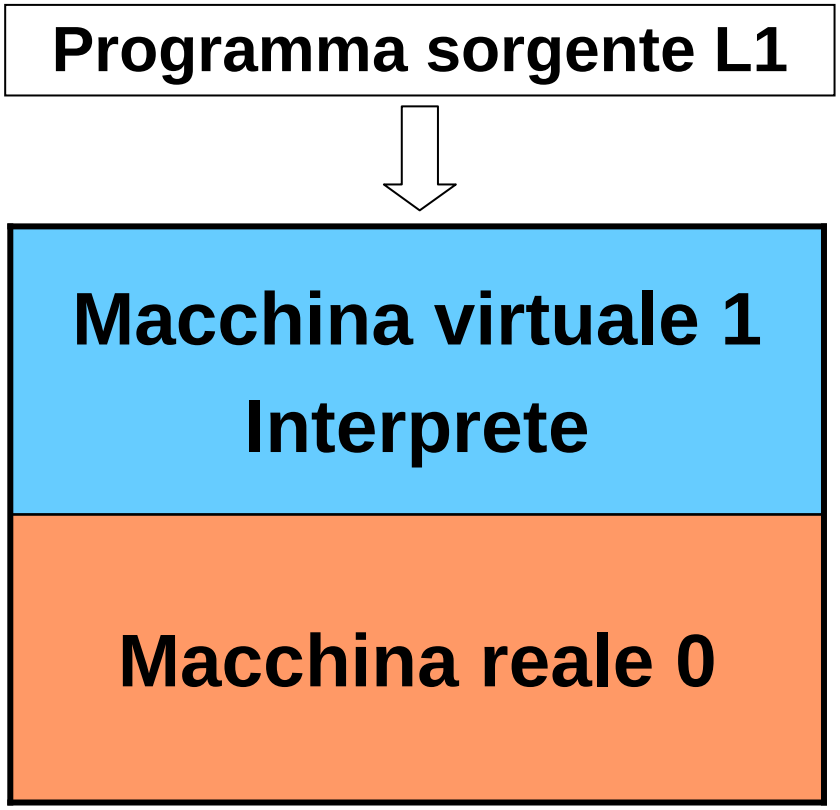
Linguaggio di livello n

- Servono linguaggi più vicini al programmatore e al tipo di problema da risolvere
- Il codice deve essere indipendente dall'architettura ISA e quindi portabile
- Ad un livello inferiore il linguaggio deve essere interpretato da una macchina.
  - Si hanno 3 casi:
    - Interpretazione
    - Compilazione
    - Approccio ibrido

Interpretazione

- Ogni istruzione viene letta, decodificata ed eseguita immediatamente dall'interprete diventando una sequenza di istruzioni del livello inferiore
- L'interprete può essere implementato tramite software o hardware.
- Il processore è un interprete hardware in grado di eseguire programmi in linguaggio macchina

Interpretazione

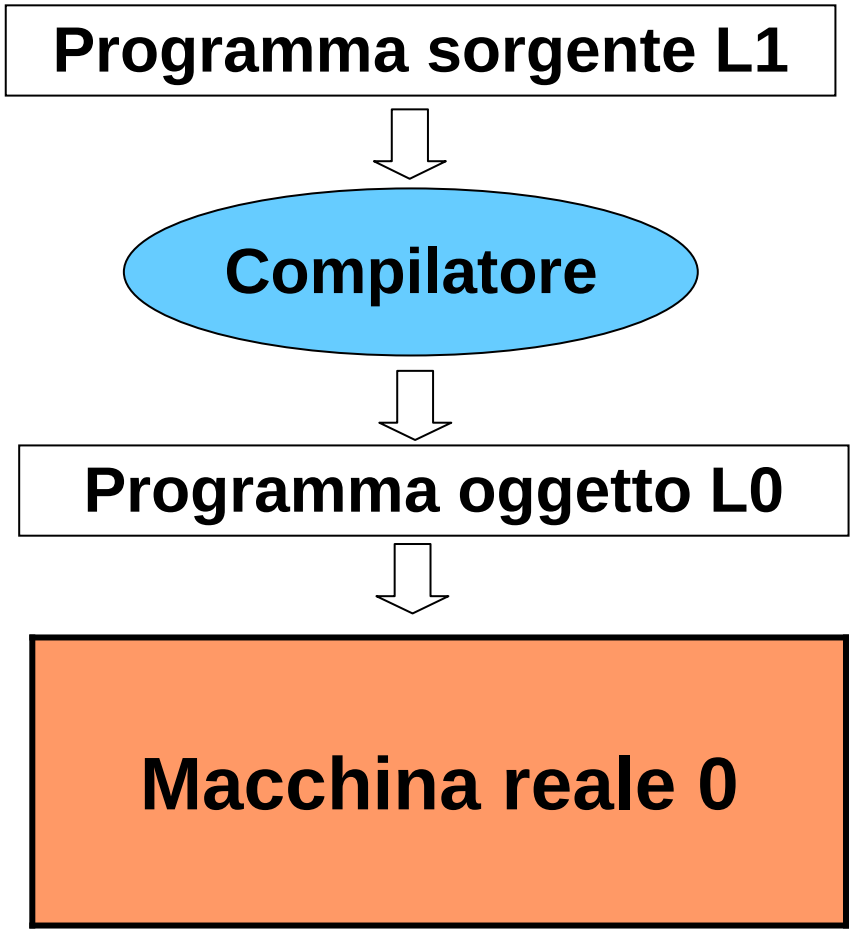


Il programma sorgente L1 viene eseguito immediatamente dall'interprete, che lo traduce in istruzioni in linguaggio macchina

Interpretazione

- È possibile avere più livelli di macchine virtuali
- Il programmatore del linguaggio L_n non è tenuto a conoscere il linguaggio L_{n-1}
- Linguaggi interpretati: Javascript, VBscript, PHP, ecc.
- Più linguaggi disponibili su una stessa piattaforma, se è disponibile l'interprete

Compilazione

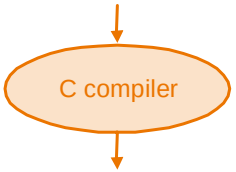


A differenza dell'interpretazione prima dell'esecuzione è necessaria una fase in più ovvero la compilazione stessa

Compilazione: esempio

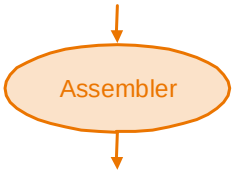
High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```



Assembly language program (for MIPS)

```
swap:
     muli $2, $5,4
     add  $2, $4,$2
     lw   $15, 0($2)
     lw   $16, 4($2)
     sw   $16, 0($2)
     sw   $15, 4($2)
     jr   $31
```



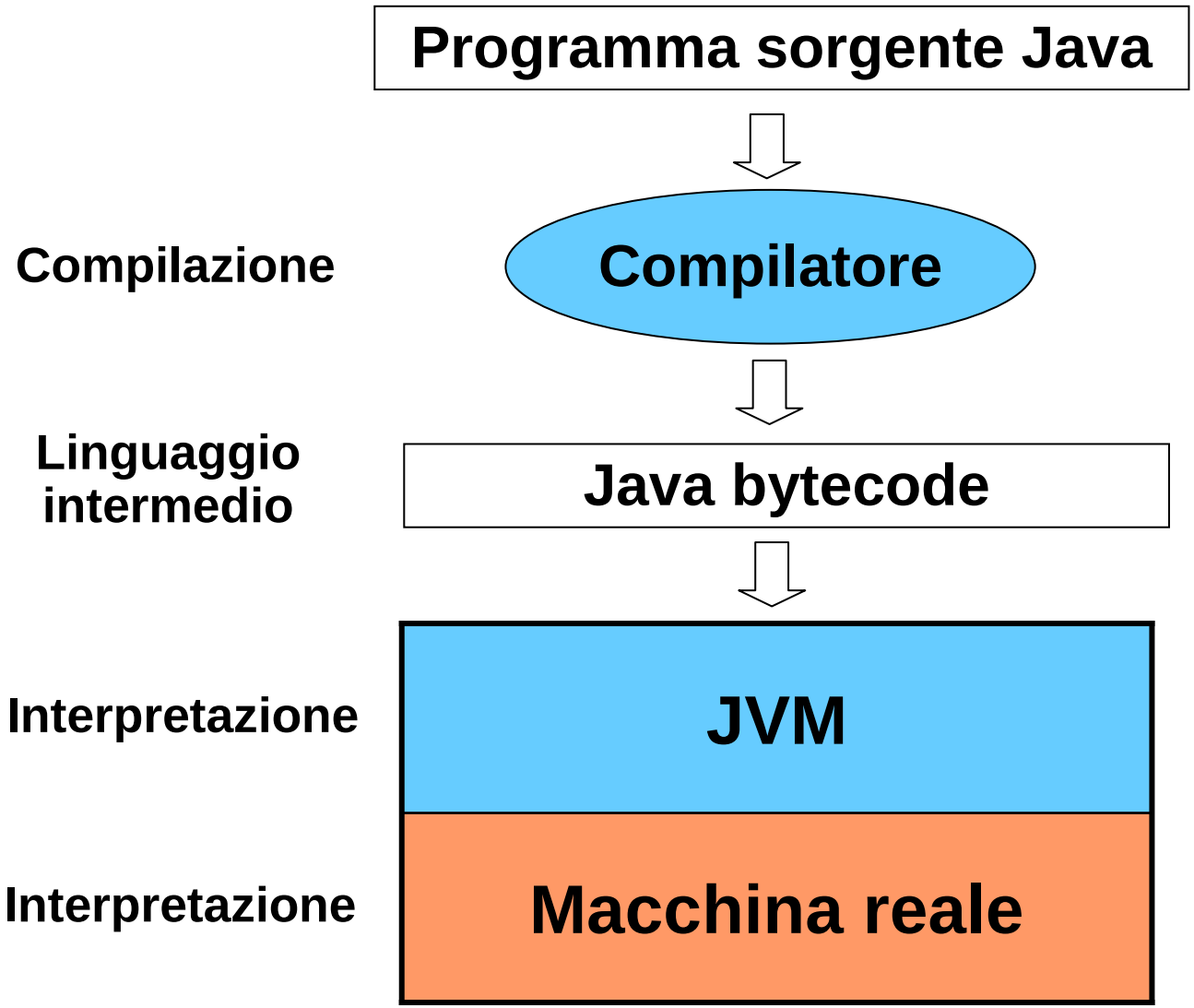
Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Compilazione

- Il compilatore traduce il linguaggio L1 nel linguaggio L0, che viene interpretato dalla macchina reale
- Il programmatore del linguaggio L1 non è tenuto a conoscere il linguaggio L0
- Linguaggi compilati (in linguaggio macchina): C, C++, Pascal, ecc.
- Più linguaggi disponibili su una stessa piattaforma, se è disponibile il compilatore

Approccio ibrido: esempio



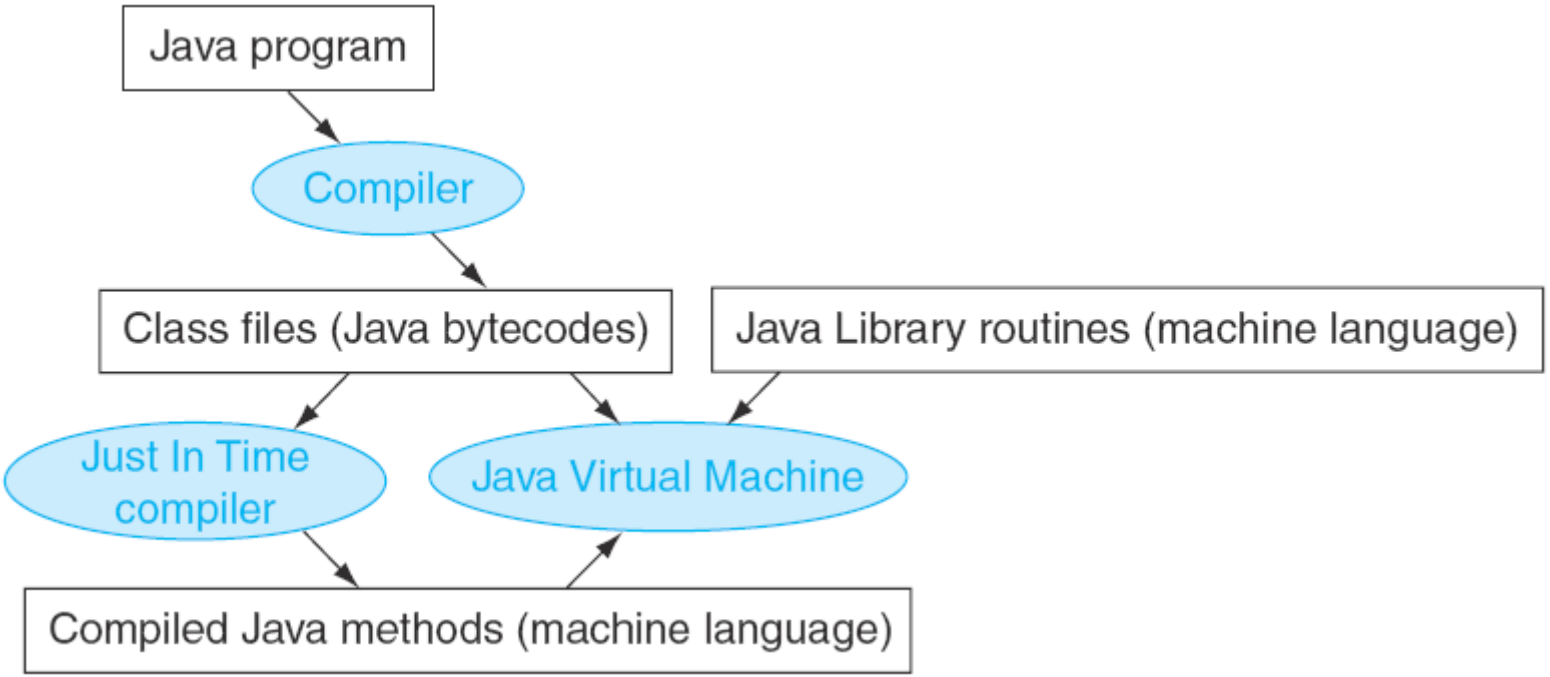
Compilazione in linguaggio intermedio

- I programma Java sono compilati in un linguaggio macchina detto **bytecode** per un'apposita macchina virtuale: la **Java Virtual Machine**
- I programma in Java possono girare su tutte le piattaforme per cui esiste una JVM (**portabilità**)
- I programmi in C possono girare solo sulla piattaforma per cui sono stati compilati (la compilazione su piattaforme diverse non è sempre immediata. Bisogna considerare anche altri fattori tra cui il sistema operativo)
- Il corrispettivo Microsoft della piattaforma Java (Sun) è il .NET Framework (Visual basic, C#). Il Microsoft Intermediate Language (MSIL) è il corrispettivo del bytecode.

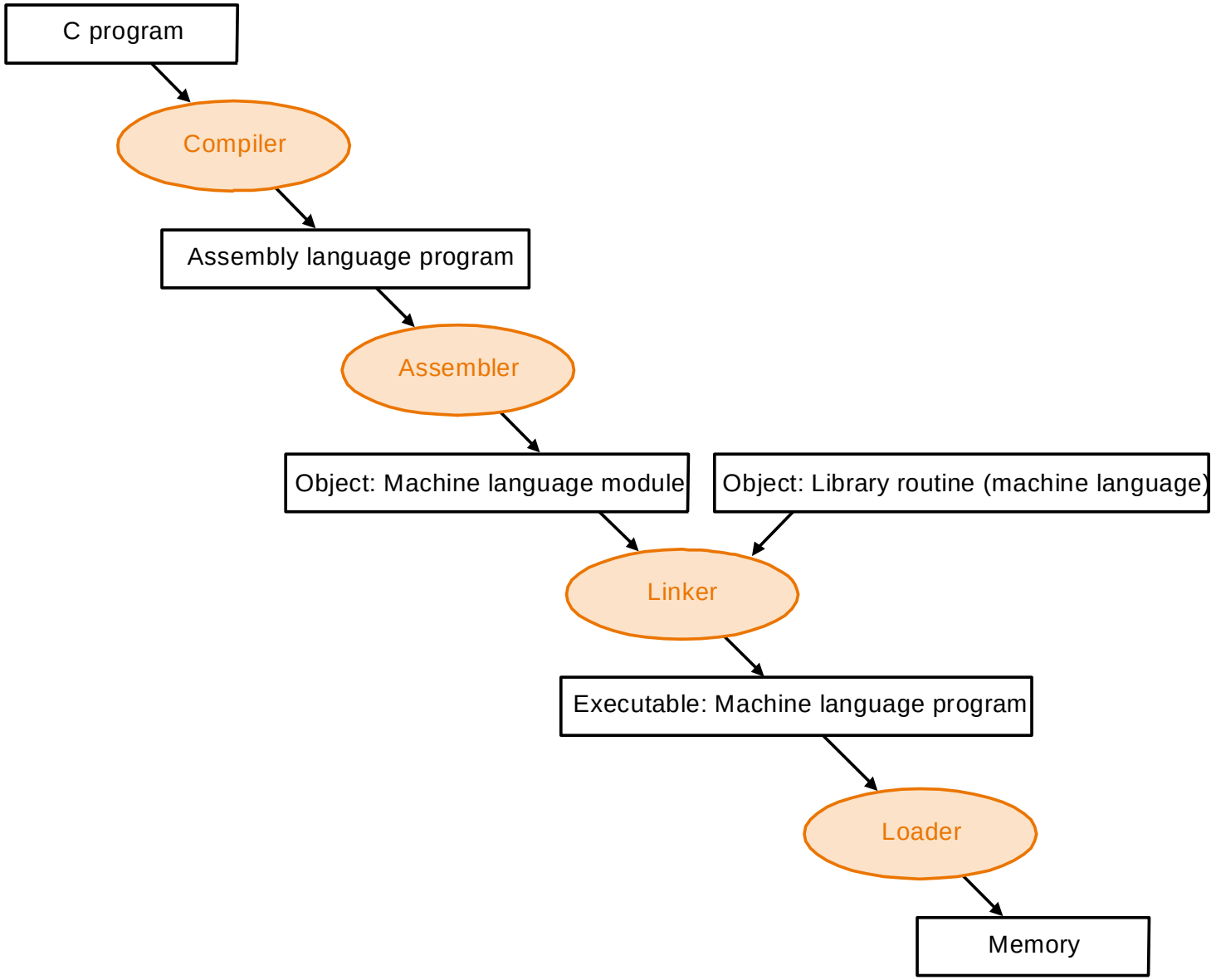
Interpretazione / Compilazione

- I programmi interpretati sono più lenti
- Compilazione **Just In Time** (JIT) per migliorare le prestazioni
- L'interprete offre un maggior controllo degli errori
- Il compilatore solo nella prima fase di compilazione - **compile-time** e non nella seconda fase di esecuzione - **run-time**.
- Con un linguaggio interpretato lo sviluppo è più rapido se la compilazione è lenta.

Compilazione JIT



Dalla compilazione all'esecuzione



Linker

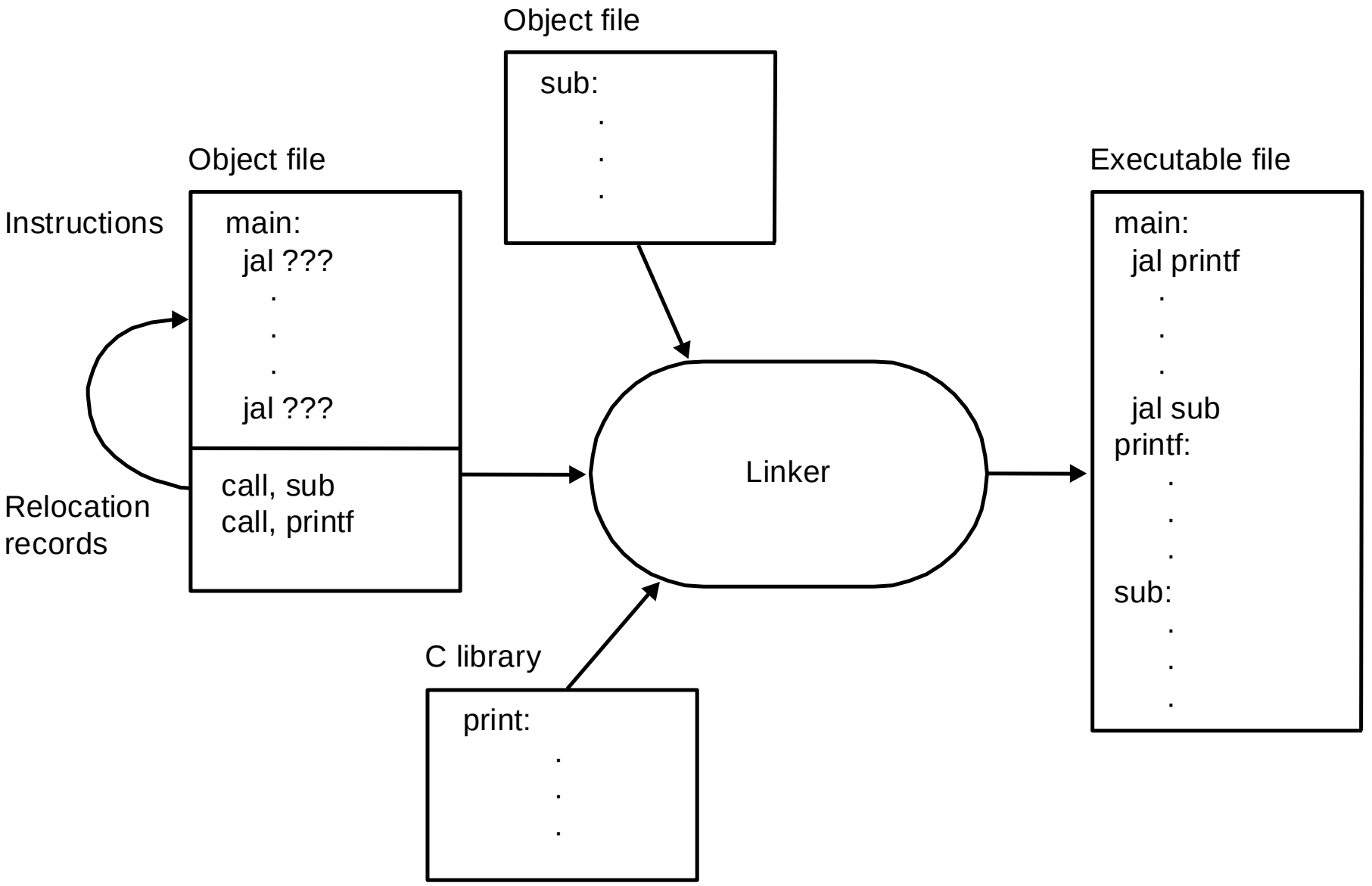
Per ottimizzare la produzione/gestione del codice sorgente, parti correlate di un programma vengono raccolte in file differenti: **moduli** / **librerie**

Compilazione separata

Il linker è lo strumento che unisce più file oggetto per produrre l'**eseguibile**

- Risolve i riferimenti esterni: cerca i moduli / librerie usati dal programma
- Mette simbolicamente in memoria codice e dati dei moduli
- **Rilocazione**: determina l'indirizzo delle etichette relative a istruzioni e dati (informazioni di rilocazione per *aggiustare* i riferimenti assoluti)

Linker


Object file
sub:
.
.
.
Object file
Instructions
main:
jal ???
.
.
.
jal ???
call, sub
call, printf
Relocation
records
Linker
Executable file
main:
jal printf
.
.
.
jal sub
printf:
.
.
.
sub:
.
.
.
C library
print:
.
.
.

Loader

Il file eseguibile prodotto dal linker deve essere caricato in memoria principale (es.: dall'hard disk) ed eseguito.

Il Sistema Operativo ha questo compito.

Tra le varie attività:

- Allocazione di un appropriata quantità di memoria
- Passaggio dei parametri

Il SO consiste spesso in un insieme di file su un supporto di memorizzazione.

Il sistema di archiviazione (File System) di tali file è noto solo al SO stesso.

Chi carica il SO?

Bootstrapping / Booting

È il processo che fa partire il sistema operativo.

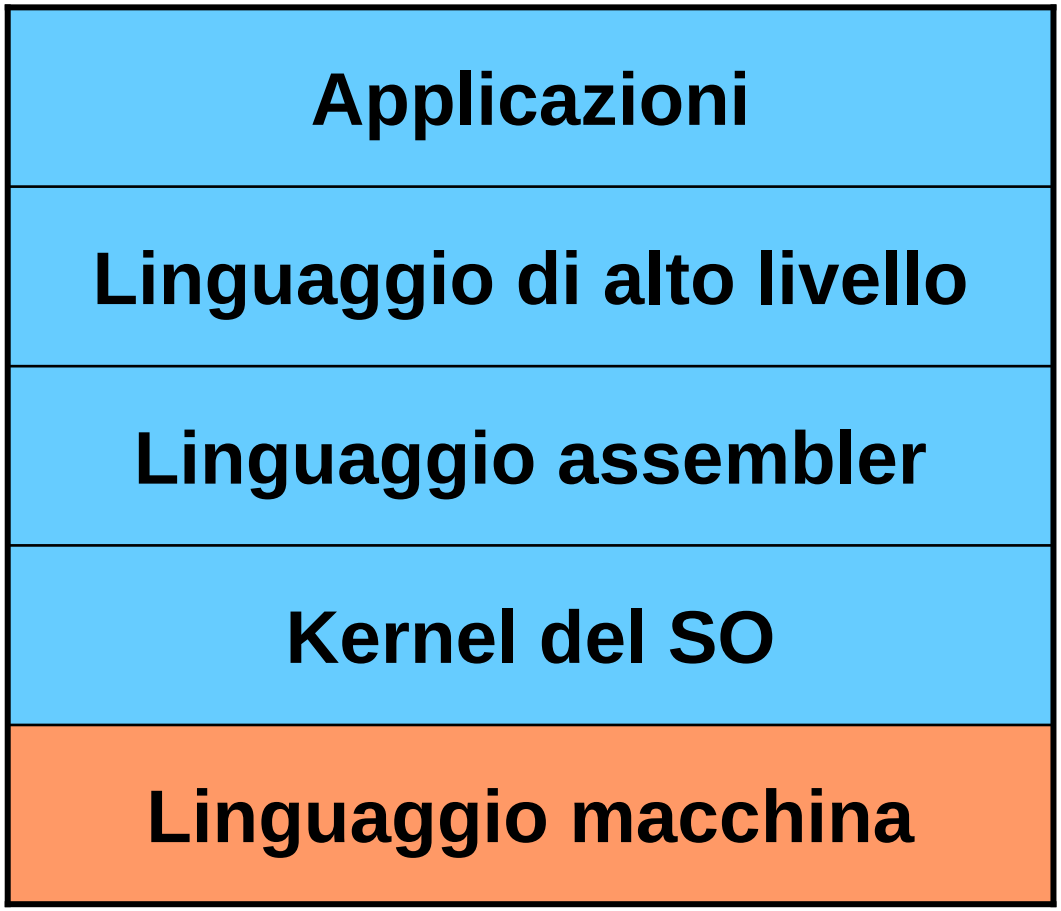
In una ROM o memoria flash è memorizzato il **BIOS** (Basic Input / Output System) un firmware eseguito dalla CPU all'accessione, che svolge le funzioni seguenti:

- POST (Power On Self Test)
- Accesso al dispositivo di boot (Es.: hard disk, CDROM, floppy, USB drive)
- Caricamento del settore di boot ed esecuzione del **boot loader** (NTLDR, LILO, ecc.)

Il boot loader si occupa di caricare ed eseguire il **kernel** del sistema operativo, che completa le operazioni successive.

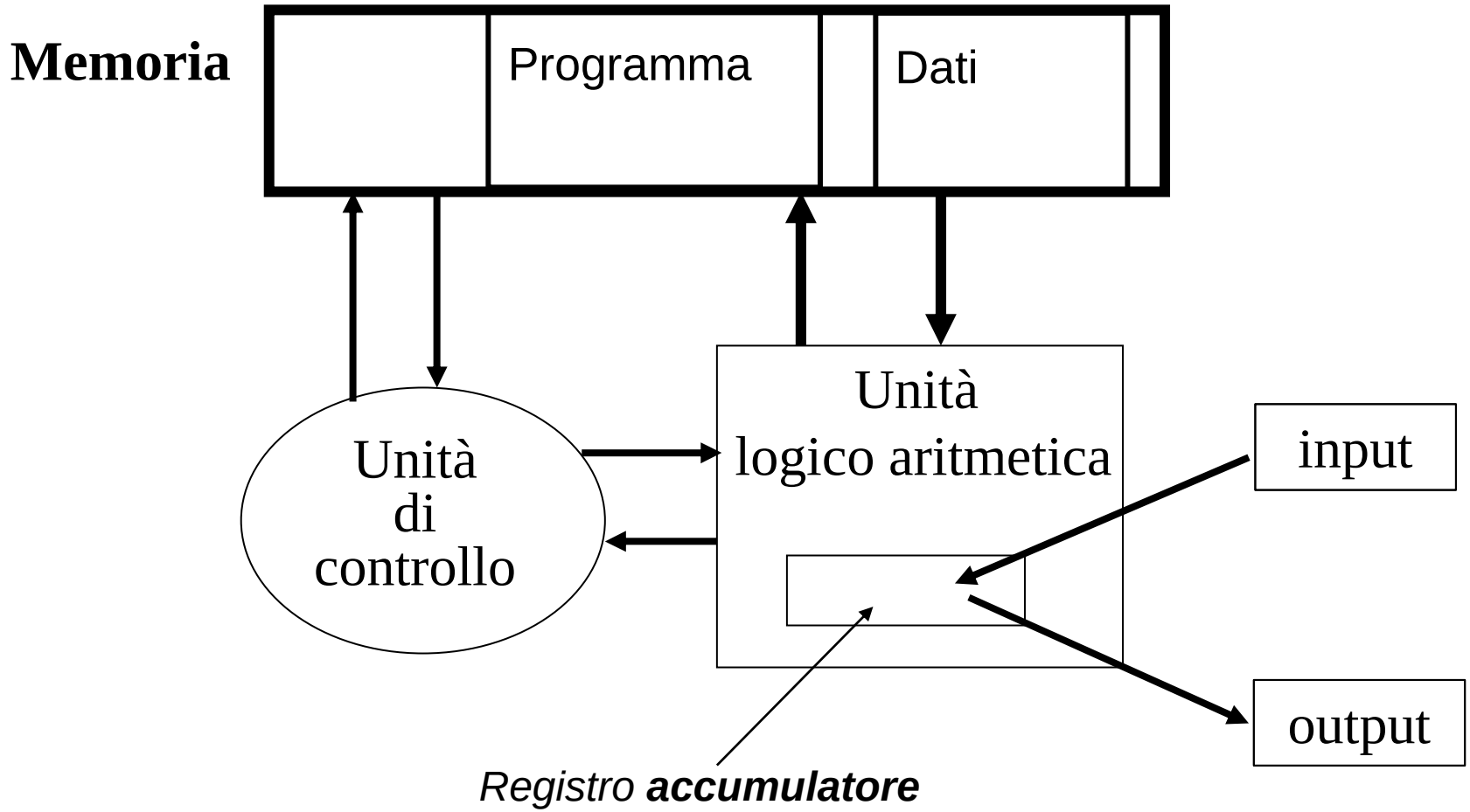
Livelli di astrazione

Utente



Il modello di Von Neumann

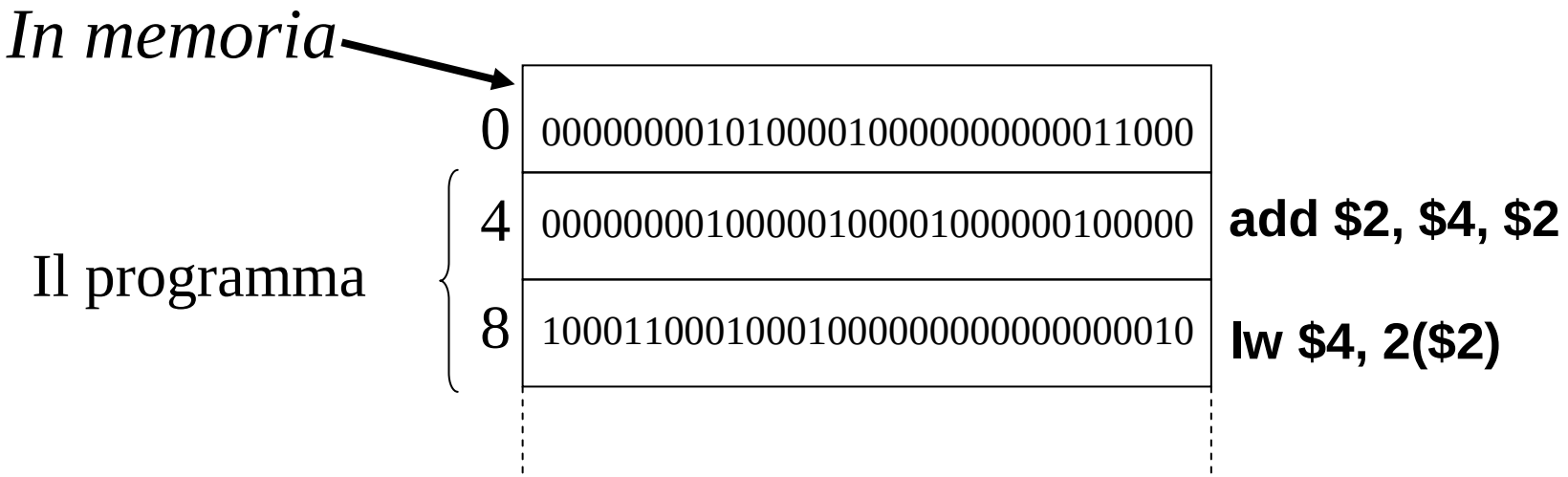
Stored-program computer



Il modello di Von Neumann

- La partecipazione di Von Neumann al progetto dell'ENIAC (~1946), il primo calcolatore elettronico "general-purpose", portò alla realizzazione dell'EDSAC (1949), il primo calcolatore elettronico "general-purpose" a **programma memorizzato**.
- Gran parte dei calcolatori attuali sono un'evoluzione di questa struttura
- Operazioni principali: **FETCH** – **DECODE** – **EXECUTE**
- Esecuzione sequenziale (nessun parallelismo)

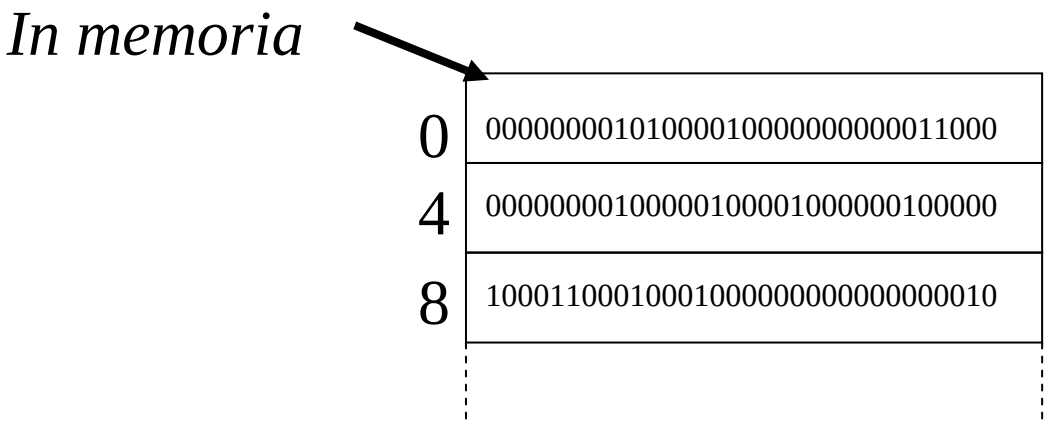
Un esempio di due istruzioni



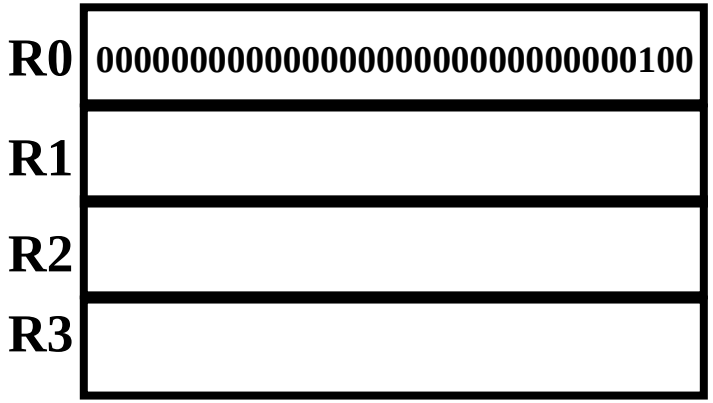
Somma il contenuto del registro R_2 al contenuto di R_4 e metti il risultato in R_2.

Poi carica nel registro R_4 il valore della parola presente all'indirizzo di memoria ottenuto sommando 2 al contenuto in R_2

Un esempio di due istruzioni



Anche i registri contengono dati



Passi per eseguire l'addizione

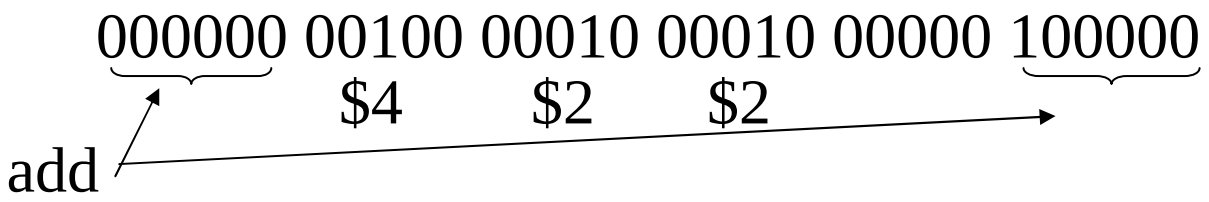
- Passo 1: Carica istruzione in IR e aggiorna PC

  IR ← 00000000100000100001000000100000

  (si indica di solito con IR ← [PC])

  R_A ← [PC] R_B ← [R0]= 4

- Passo 2: DEC decodifica istruzione in IR



ALU calcola [PC] +4 e lo memorizza in R_{ACC}

Passi per eseguire l'addizione

- Passo 3: Caricamento valori dei registri

  $R_A \leftarrow R_2$

  $R_B \leftarrow R_4$

  Si aggiorna il PC $\leftarrow R_{ACC}$

- Passo 4: Somma (operazione con ALU)

  $R_{ACC} \leftarrow R_A + R_B$

- Passo 5: Memorizza risultato in R_2

  $R_2 \leftarrow R_{ACC}$

  TOTALE = 5 passi (eseguiti in 5 cicli di clock)

Segnale di clock

- È caratterizzato da una frequenza costante e sincronizza i vari eventi all'interno dell'hardware
- Determina in questo modo intervalli di tempo discreti che sono denominati cicli di clock
- Si fa riferimento alla durata di un periodo di clock oppure alla frequenza di clock

Passi per eseguire una load

- Passo 1: Carica istruzione in IR e aggiorna PC

  $IR \leftarrow (PC)$

  $R_A \leftarrow [PC]$ $\qquad\qquad$ $R_B \leftarrow [R0]= 4$

- Passo 2: Decodifica istruzione in IR

  100011 00010 00100 0000000000000010

  lw $2 $4 offset= 2

  ALU calcola [PC] +4 e lo memorizza in R_{ACC}

- Passo 3: Copia 2 e R_2 nei registri usati dalla ALU

  $R_A \leftarrow 2$

  $R_B \leftarrow R_2$

Si aggiorna il PC $\leftarrow R_{ACC}$

- Passo 4: Somma contenuto registri

  $R_{ACC} \leftarrow R_A + R_B$
- Passo 5: Copia contenuto di R_{ACC} in MAR

  $MAR \leftarrow R_{ACC}$
- Passo 6: Accedi alla memoria e carica dato

  $MDR \leftarrow (MAR)$
- Passo 7: Copia contenuto di MDR in R_4

  $R_4 \leftarrow MDR$

TOTALE = 7 passi (eseguiti in 7 cicli di clock – bisogna considerare anche l'accesso in memoria)

Riferimenti

Computer Organization and Design

The Hardware/Software Interface 3rd Edition

David A. Patterson, John L. Hennessy

Capitolo 1 e parte del capitolo 2

Appendice A

Versione italiana:

Struttura e Progetto dei Calcolatori

L'Interfaccia Hardware-Software

2ª edizione Zanichelli

http://en.wikipedia.org/ o http://it.wikipedia.org/