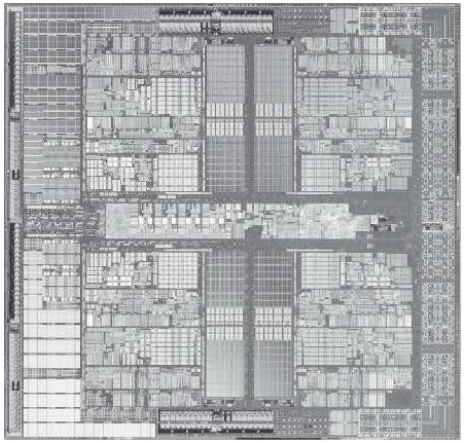
Università degli Studi di Cassino



Corso di Calcolatori Elettronici

Introduzione

Anno Accademico 2010/2011

Francesco Tortorella

Contenuti del corso

- Misura delle prestazioni
- Modello di programmazione del processore
- Programmazione in linguaggio assembly
- Rappresentazione dei dati
- Elementi di progettazione logica dei circuiti
- Architettura del processore
- Sistema di bus
- Gerarchia di memoria
- Sistema di I/O

Testo adottato:

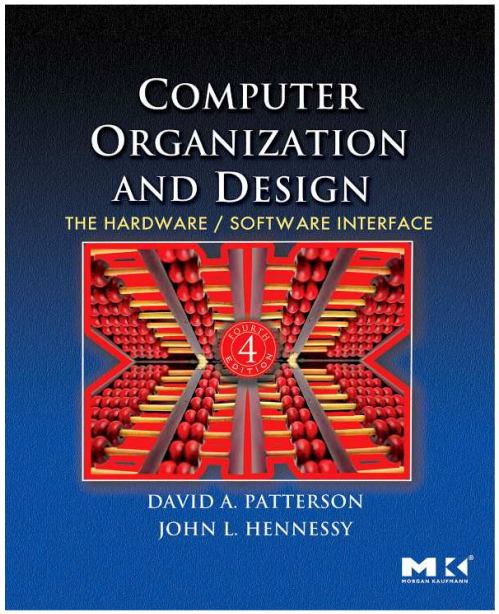
Patterson, Hennessy

Computer Organization and Design

The hardware/software interface

4th edition

Morgan Kaufmann 2009



Versione italiana:

Patterson, Hennessy

Struttura e progetto dei calcolatori

Traduzione della 4a ed. inglese

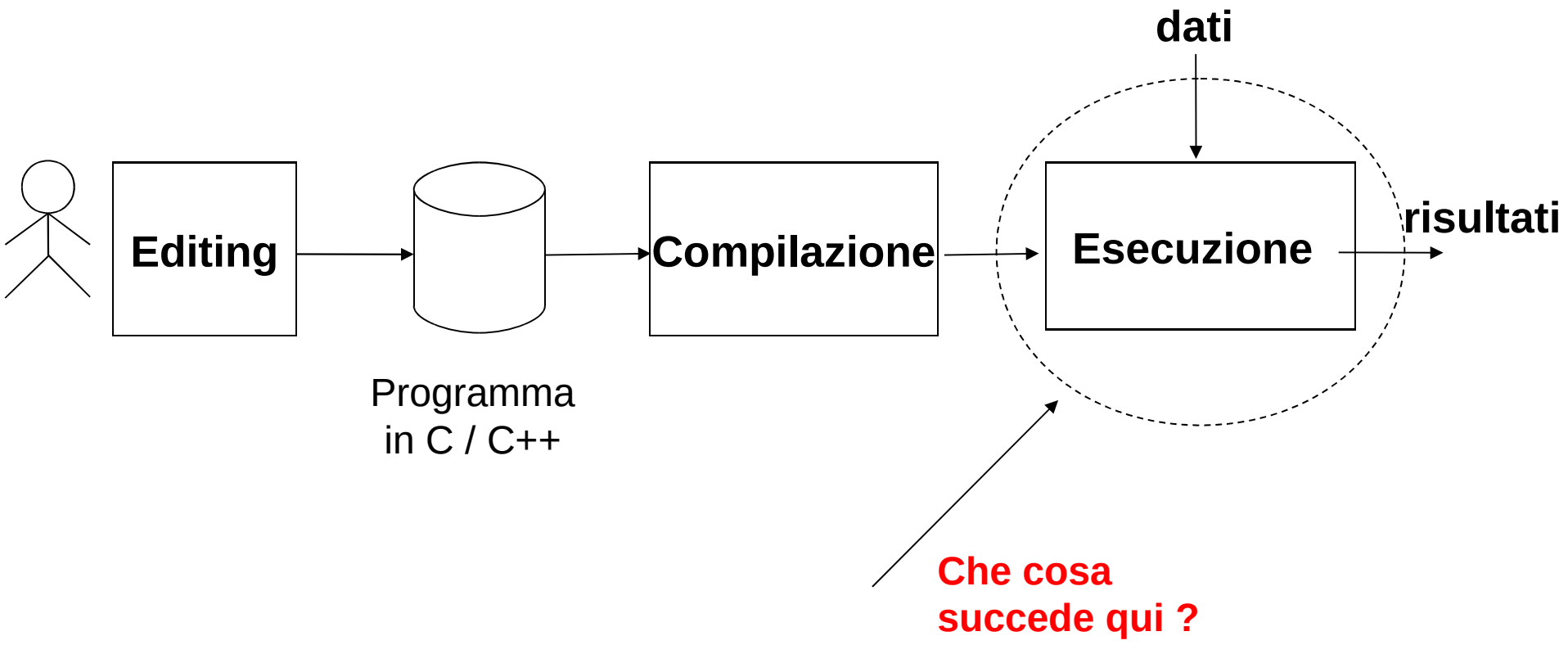
Zanichelli € 62,40

…chi siamo, da dove veniamo ?

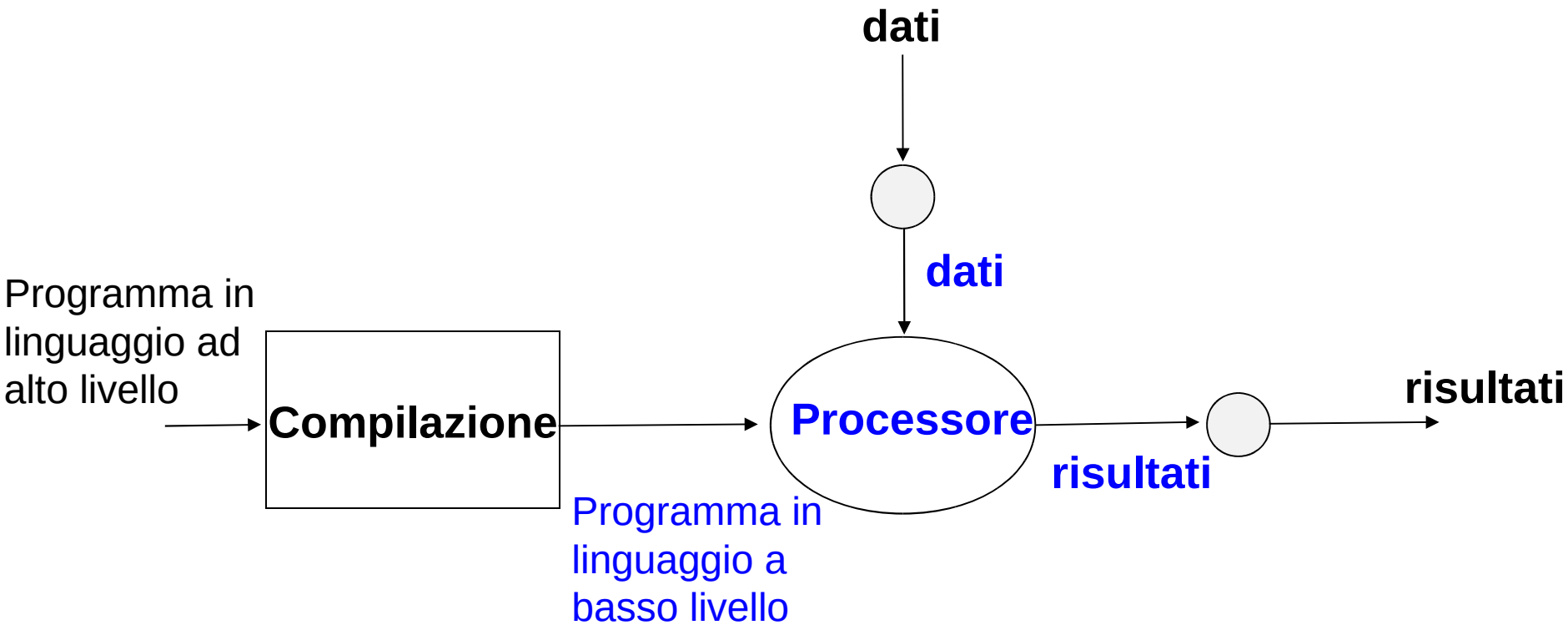
- Fondamenti di Informatica:
  - Strutture dati
  - Costrutti di programmazione
  - Algoritmi fondamentali
- Programmazione a Oggetti:
  - Strutture dati complesse
  - Ricorsione
- Basi di Dati
  - Modello Relazionale dei Dati

C / C++

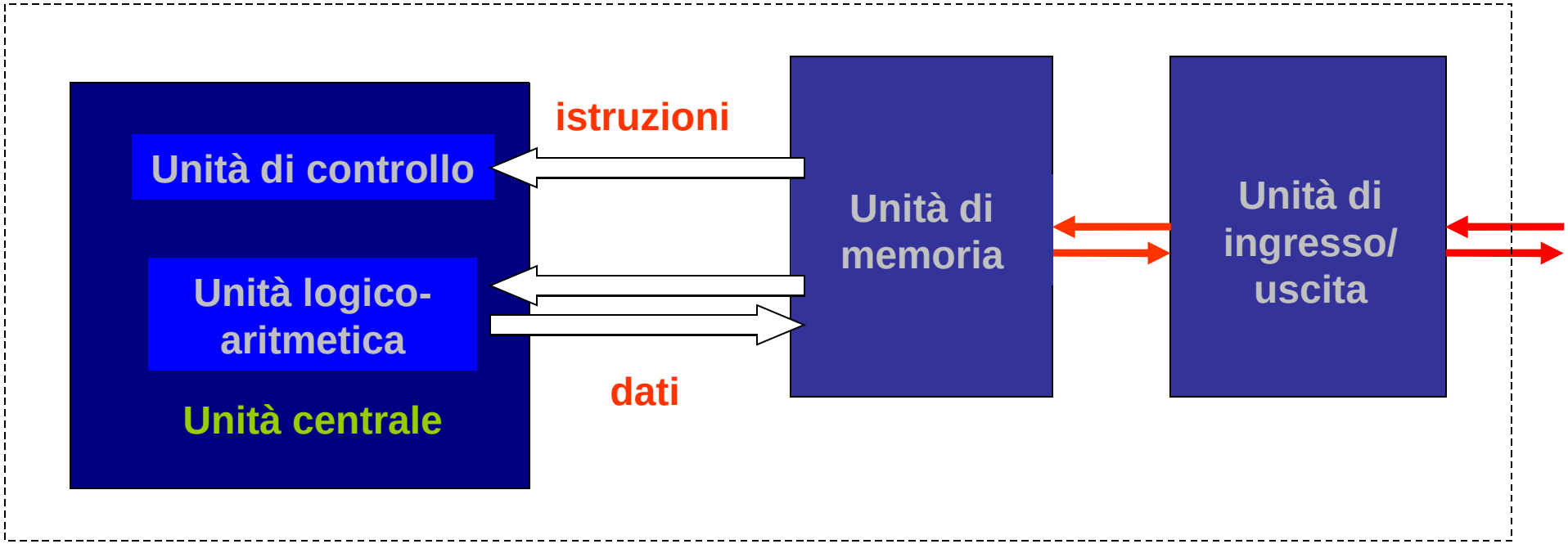
Una tipica sessione di lavoro



Che cosa succede sotto il coperchio ?



Organizzazione del calcolatore

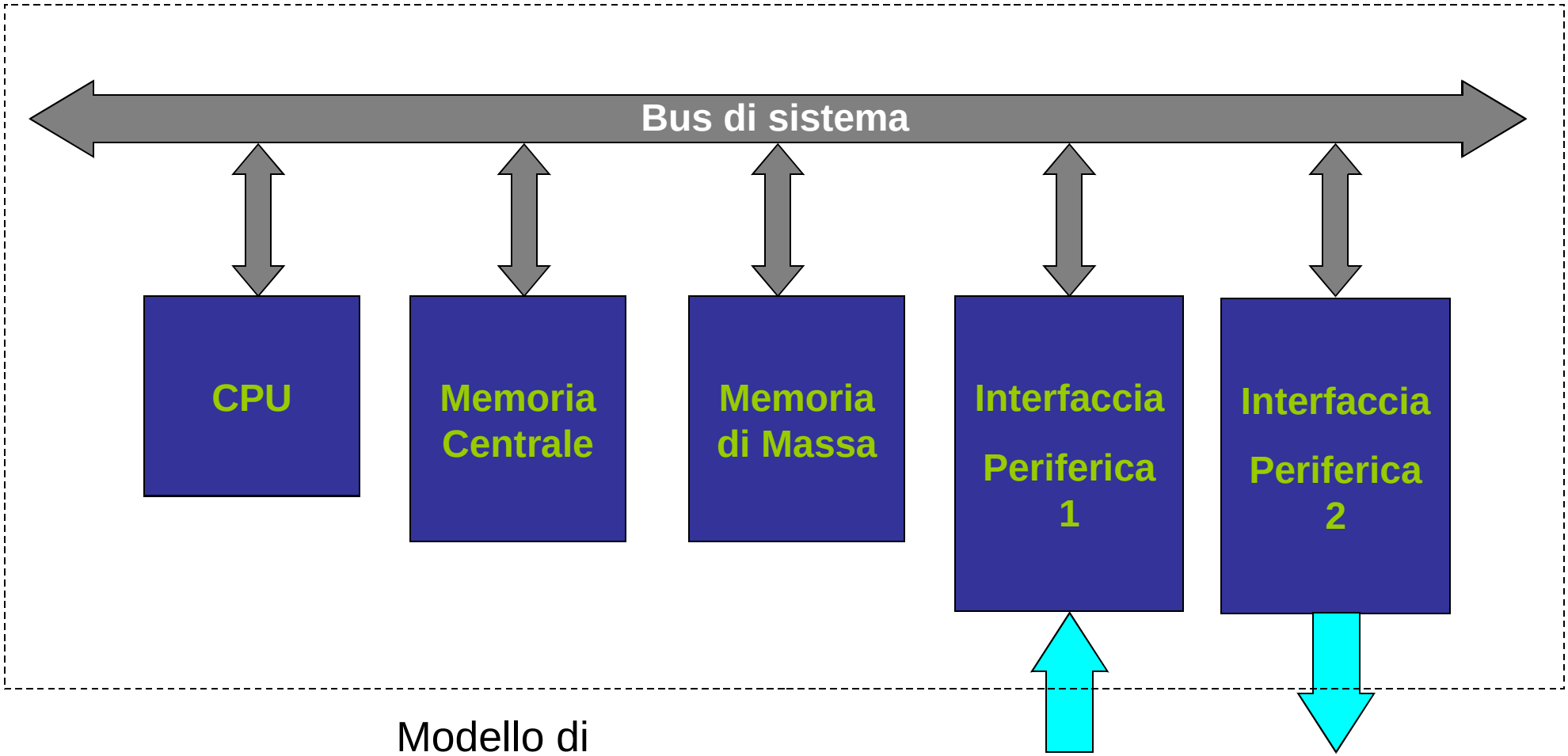


Modello logico

singole componenti

flussi di dati e istruzioni

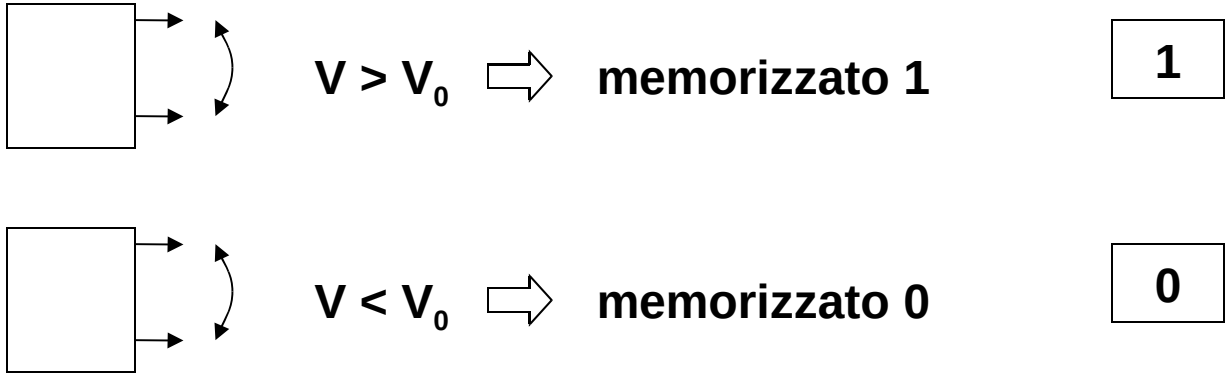
Modello di von Neumann



Modello di implementazione

La memorizzazione dei dati e delle istruzioni

- La più piccola unità di informazione memorizzabile (e quindi utilizzabile) è il **bit**, che può assumere valore 0 o 1.
- Il dispositivo utilizzato per memorizzare un bit è un **elemento bistabile**, cioè un dispositivo elettronico che può assumere uno tra due stati stabili (es. due livelli differenti di tensione), ognuno dei quali viene fatto corrispondere a 0 o a 1 (**cella di memoria**).



Operazioni possibili su una cella di memoria

Scrittura

- La cella di memoria viene caricata con un determinato valore che permane memorizzato finchè:
  - la cella viene alimentata elettricamente
  - non si esegue un'altra operazione di scrittura che modifica il valore precedentemente memorizzato

Lettura

- Si accede alla cella di memoria per consultarne il valore e copiarlo su un'altra cella di memoria.

Il registro di memoria

- Un insieme di N celle elementari può assumere uno tra 2^N stati possibili
- Un tale insieme è organizzato in un **registro di memoria**
- Il registro costituisce un supporto per la memorizzazione di un'informazione che può assumere uno tra 2^N valori possibili. In particolare un insieme di 8 bit forma un **byte**
- Sul registro sono possibili operazioni di lettura e scrittura che interessano contemporaneamente tutte le celle di memoria contenute nel registro

Il problema della codifica

- Un calcolatore può trattare diversi tipi di dati: numeri (interi, reali), testo, immagini, suoni, ecc. che vanno comunque memorizzati su registri di memoria.

- È quindi necessario adottare una **codifica** del tipo di dato considerato: occorre, cioè,

  mettere in <u>corrispondenza biunivoca</u> i valori del tipo con gli stati che può assumere il registro.

Esempio

registro da un byte $\Rightarrow 2^8 = 256$ stati possibili.
Che cosa è possibile codificare ?

Numeri naturali [0,255]

0 $\leftrightarrow$ 00000000
1 $\leftrightarrow$ 00000001
....
255 $\leftrightarrow$ 11111111

Numeri interi [-128,127]

-128 $\leftrightarrow$ 00000000
-127 $\leftrightarrow$ 00000001
0 $\leftrightarrow$ 10000000
+127 $\leftrightarrow$ 11111111

Numeri reali [0,1[

0.0000 $\leftrightarrow$ 00000000
0.0039 $\leftrightarrow$ 00000001
0.0078 $\leftrightarrow$ 00000010
....
0.9961 $\leftrightarrow$ 11111111

Caratteri

A $\leftrightarrow$ 01000001
a $\leftrightarrow$ 01100001

0 $\leftrightarrow$ 00110000
1 $\leftrightarrow$ 00110001

La codifica implica una rappresentazione dei dati <u>limitata</u> e <u>discreta</u>

Codifica delle istruzioni

Oltre ai dati, è necessario memorizzare anche le **istruzioni**, cioè le singole azioni elementari che l'unità centrale può eseguire.

Nello specificare un'istruzione, bisogna precisare l'**operazione** da compiere e i **dati** coinvolti nell'operazione.

Esempio:

dati

operazione **`somma 3 e 4`**

Come rappresentare le operazioni ?

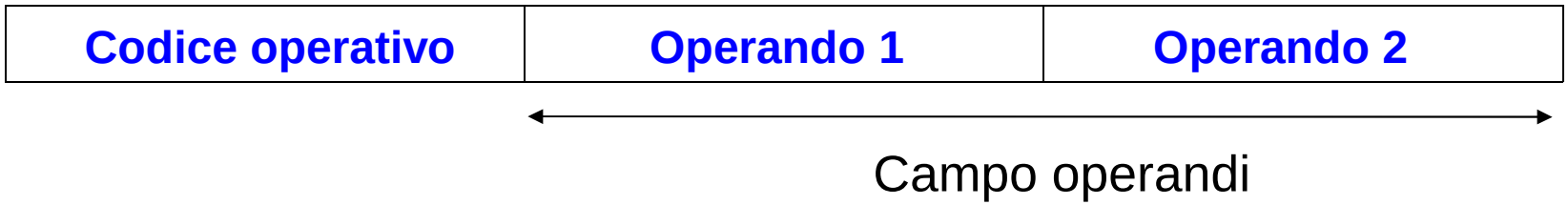
L'insieme delle diverse operazioni che l'unità centrale è in grado di eseguire è **finito** e quindi è possibile codificarlo con un certo numero di bit (**codice operativo**).

```
somma      0000
sottrai    0001
moltiplica 0010
dividi     0011
```

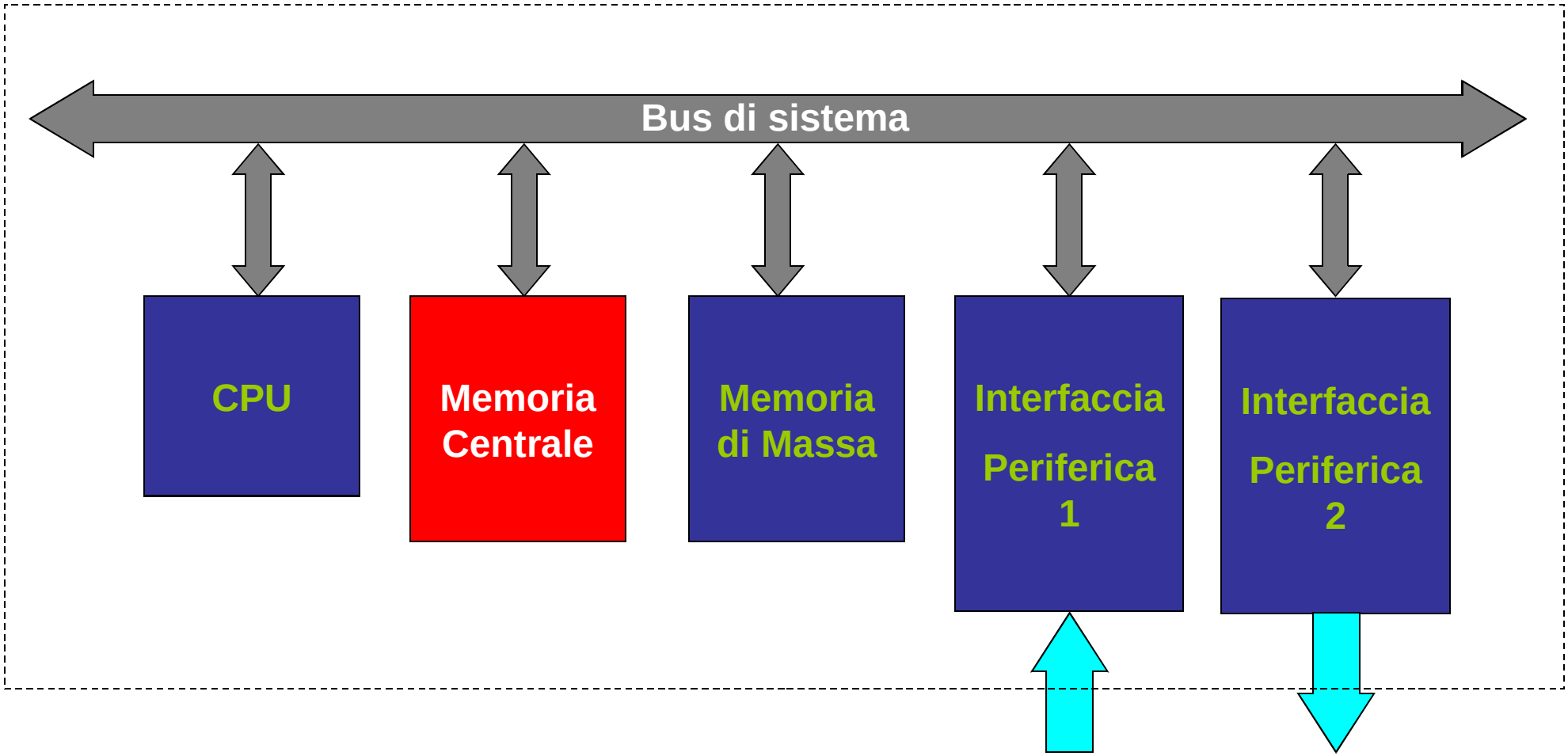
Codifica delle istruzioni

Una istruzione sarà quindi rappresentabile da una sequenza di bit divisa in due parti:

- **un codice operativo**
- **un campo operandi (1, 2 o più operandi)**



Modello di von Neumann



Organizzazione della memoria principale

La memoria principale è organizzata come un insieme di registri di uguale dimensione, ognuno dei quali è identificato tramite un numero progressivo ad esso associato, detto **indirizzo**.

Il contenuto dei registri non è immediatamente riconoscibile: non c'è distinzione esplicita tra istruzioni e dati e tra dati di tipo diverso.

Una istruzione o un dato possono risiedere su più registri consecutivi, se la dimensione del registro di memoria non è sufficiente.

Il parallelismo di accesso è definito dall'ampiezza del registro

0	**01101101**
1	**10010110**
2	**00111010**
3	**11111101**
	⋮
1022	**00010001**
1023	**10101001**

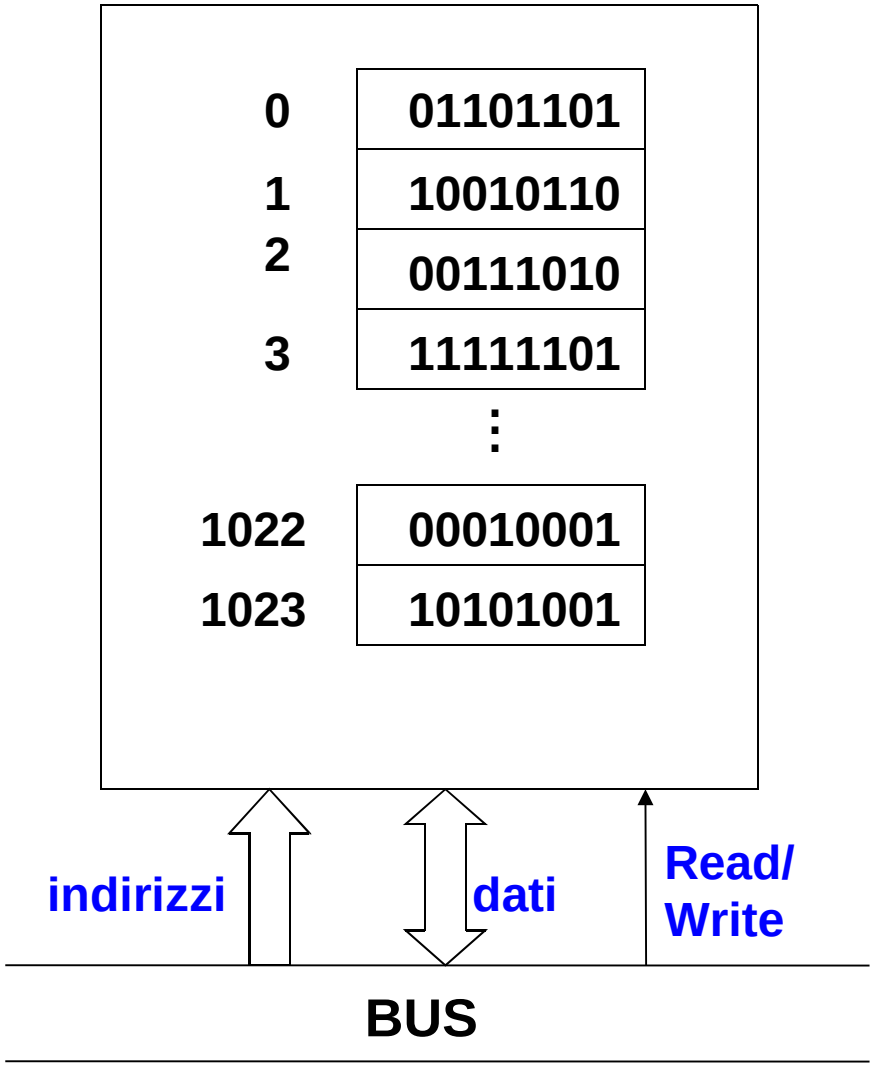
Quanti bit sono necessari per codificare un indirizzo ?

Organizzazione della memoria principale

Il modulo di memoria principale è connesso al resto del sistema tramite il BUS.

In particolare, sono presenti tre gruppi di linee:

- **linee indirizzi**
- **linee dati**
- **linee Read/Write**

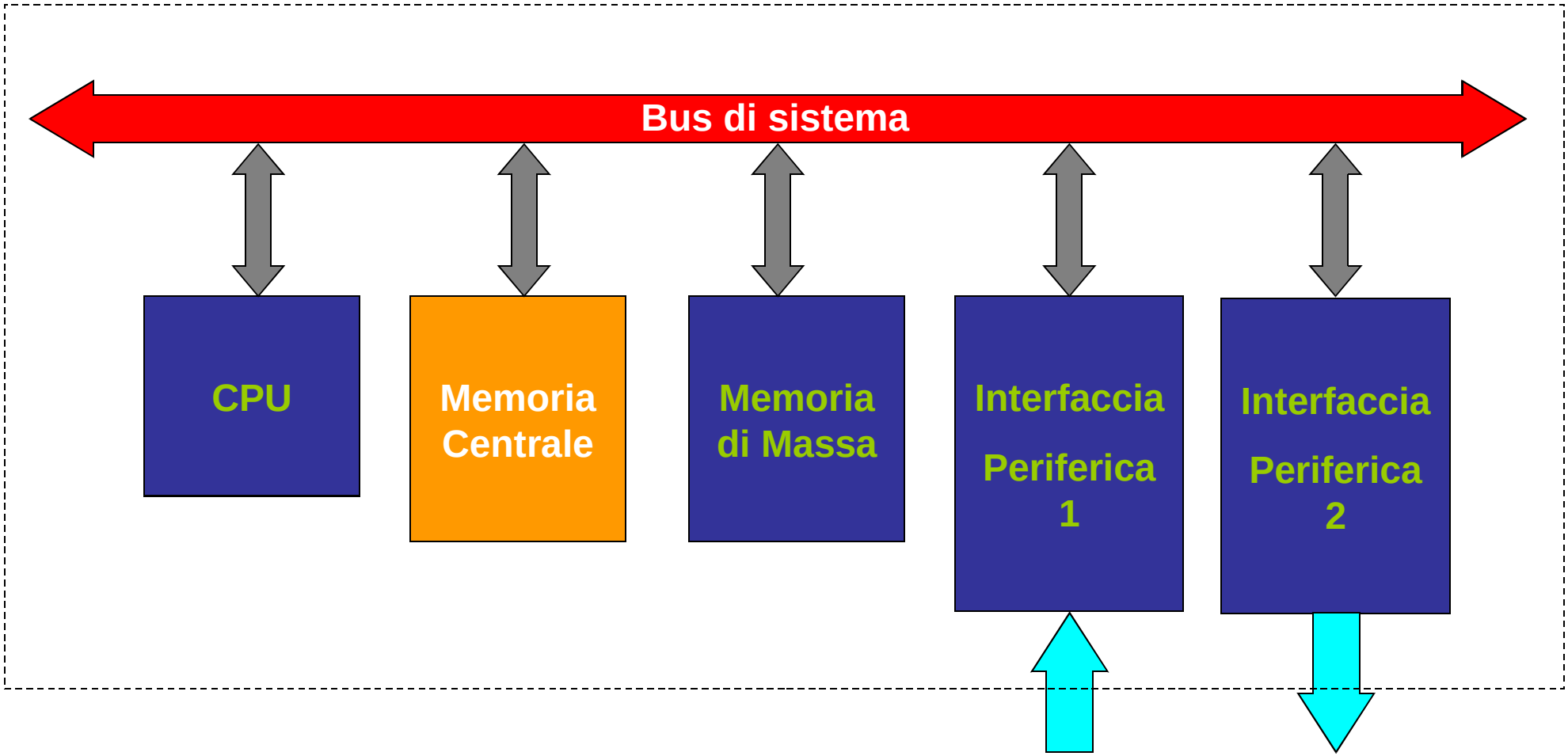


Operazioni sulla memoria principale

- Le operazioni possibili sul modulo di memoria principale sono orientate ai registri:
  - **scrittura di un valore in un registro**
  - **lettura del valore di un registro**

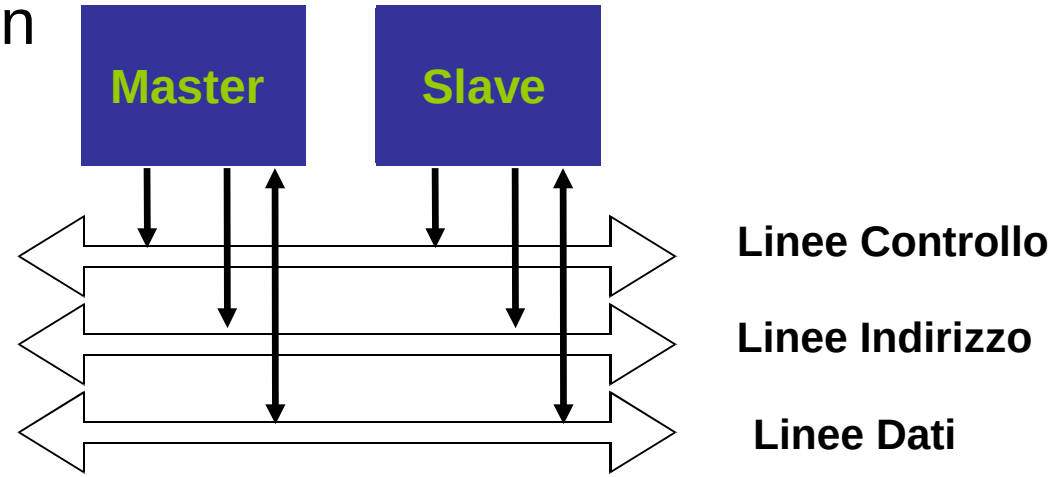
- In ogni operazione è quindi necessario specificare:
  - su quale registro si intende compiere l'operazione → **indirizzo**
  - che tipo di operazione si intende realizzare → **Read/Write**
  - in caso di scrittura, quale sia il valore da memorizzare

Modello di von Neumann

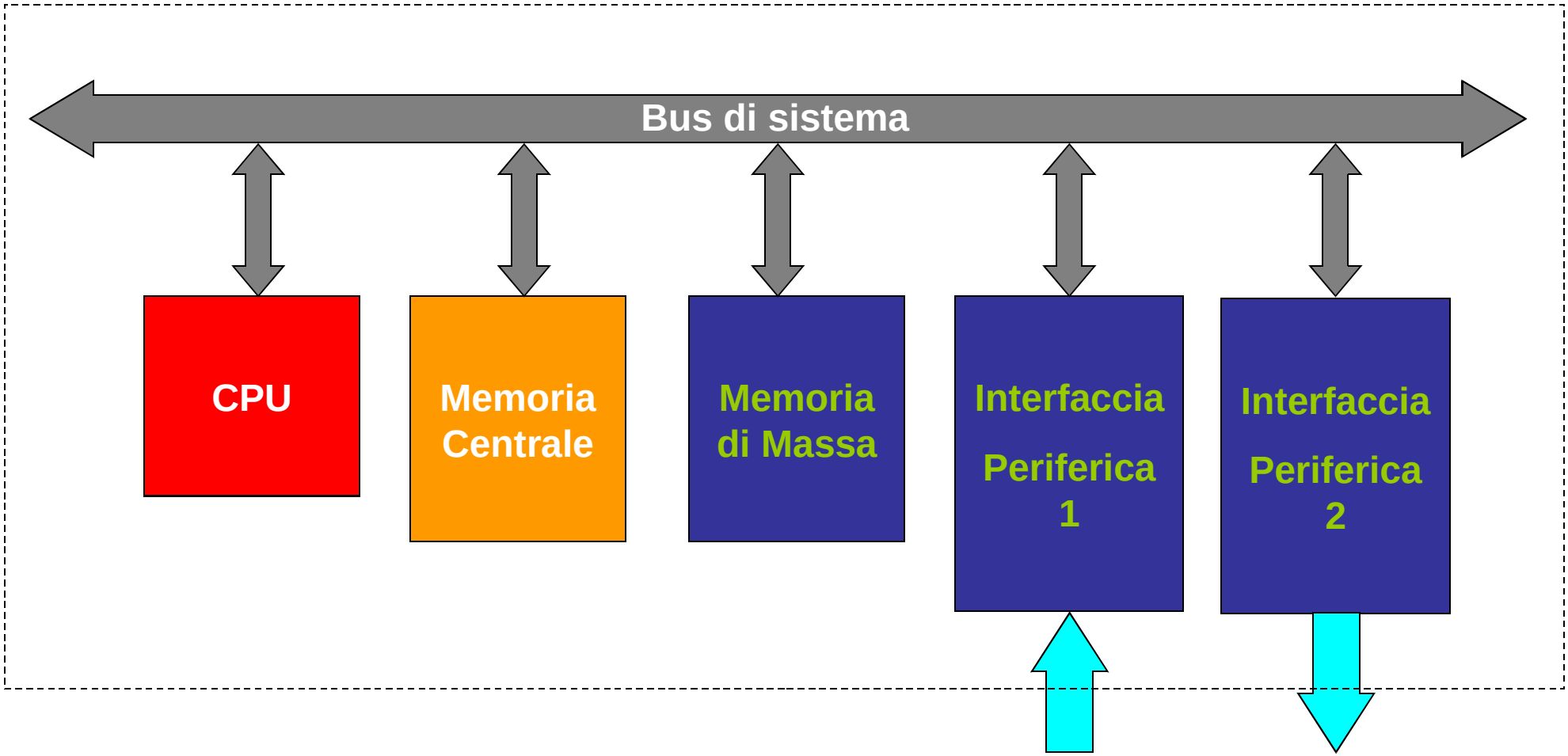


Il bus

- Forma un canale di comunicazione tra le varie unità del calcolatore.
- Tipicamente è possibile un solo colloquio alla volta tra due unità: un **master**, che ha la capacità di controllare il bus ed inizia la comunicazione, ed uno **slave**, che viene attivato dal master.
- Il bus è formato da un insieme di linee su cui viaggiano i segnali. Le linee si dividono in
  - **linee dati**
  - **linee indirizzi**
  - **linee controllo**



Modello di von Neumann

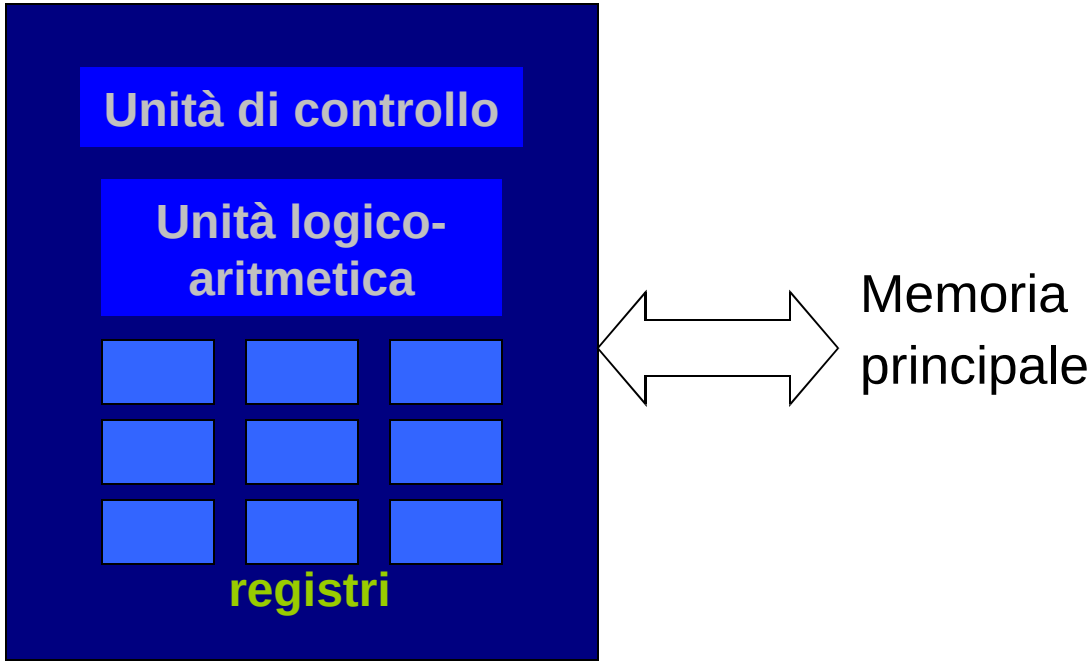


CPU (Central Processing Unit)

Funzione:
eseguire i programmi immagazzinati in memoria principale prelevando le istruzioni (e i dati relativi), interpretandole ed eseguendole una dopo l'altra

E' formata da:

- **unità di controllo**
- **unità logico aritmetica**
- **registri**



La CPU è inoltre caratterizzata dall'insieme delle istruzioni che può eseguire (instruction set)

L'Unità di controllo

E' l'unità che si occupa di dirigere e coordinare le attività interne alla CPU che portano all'esecuzione di una istruzione

Ciclo del processore

L'esecuzione di una istruzione avviene attraverso alcune fasi:

Fetch
L'istruzione da eseguire viene prelevata dalla memoria e trasferita all'interno della CPU

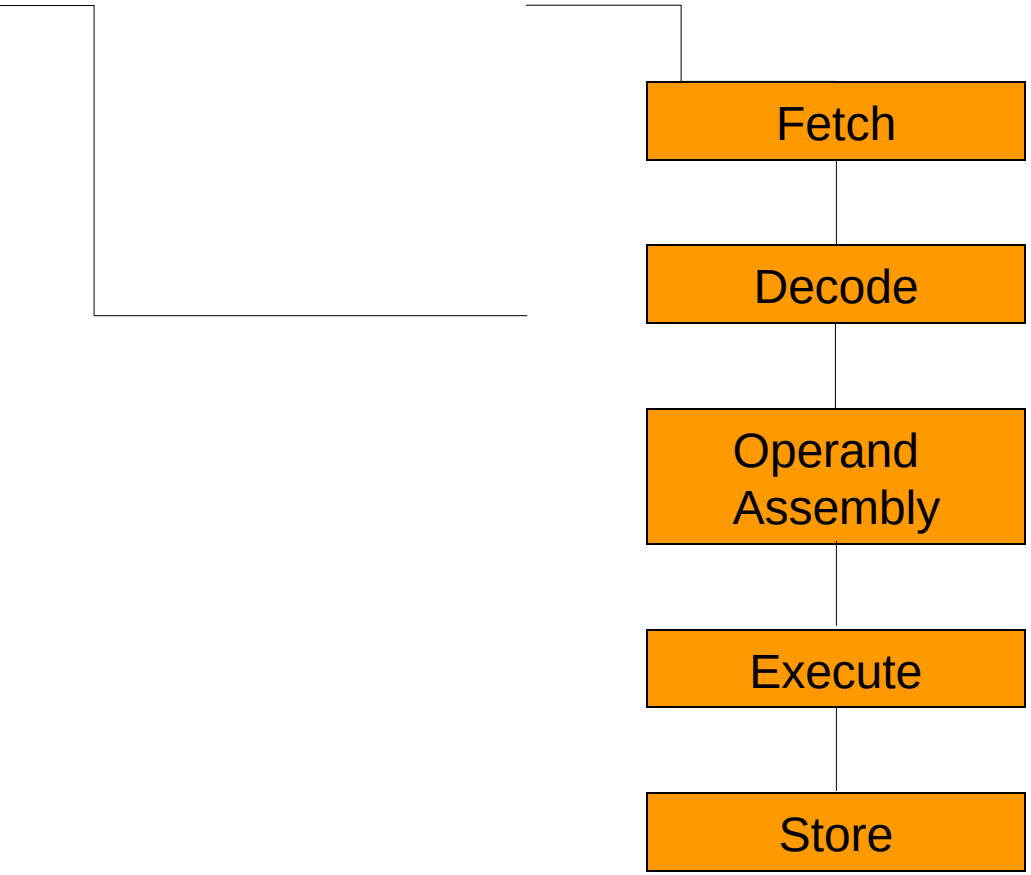
Decode
L'istruzione viene interpretata e vengono avviate le azioni interne necessarie per la sua esecuzione

Operand Assembly
Vengono prelevati dalla memoria i dati su cui eseguire l' operazione prevista dalla istruzione

Execute
Viene portata a termine l'esecuzione dell'operazione prevista dalla istruzione

Store
Viene memorizzato il risultato dell'operazione prevista dalla istruzione

L'Unità di controllo



L'unità di controllo realizza in ciclo le fasi per eseguire la sequenza di istruzioni che costituiscono il programma

L'Unità Logico Aritmetica

E' l'unità che si occupa di realizzare le operazioni logiche ed aritmetiche eventualmente richieste per eseguire un'istruzione

Operazioni Aritmetiche

ADD

SUB

MUL

DIV

REM

SET

Operazioni Logiche

CMP

AND

OR

NOT

I registri

Hanno la funzione di memorizzare all'interno della CPU dati e istruzioni necessari all'esecuzione

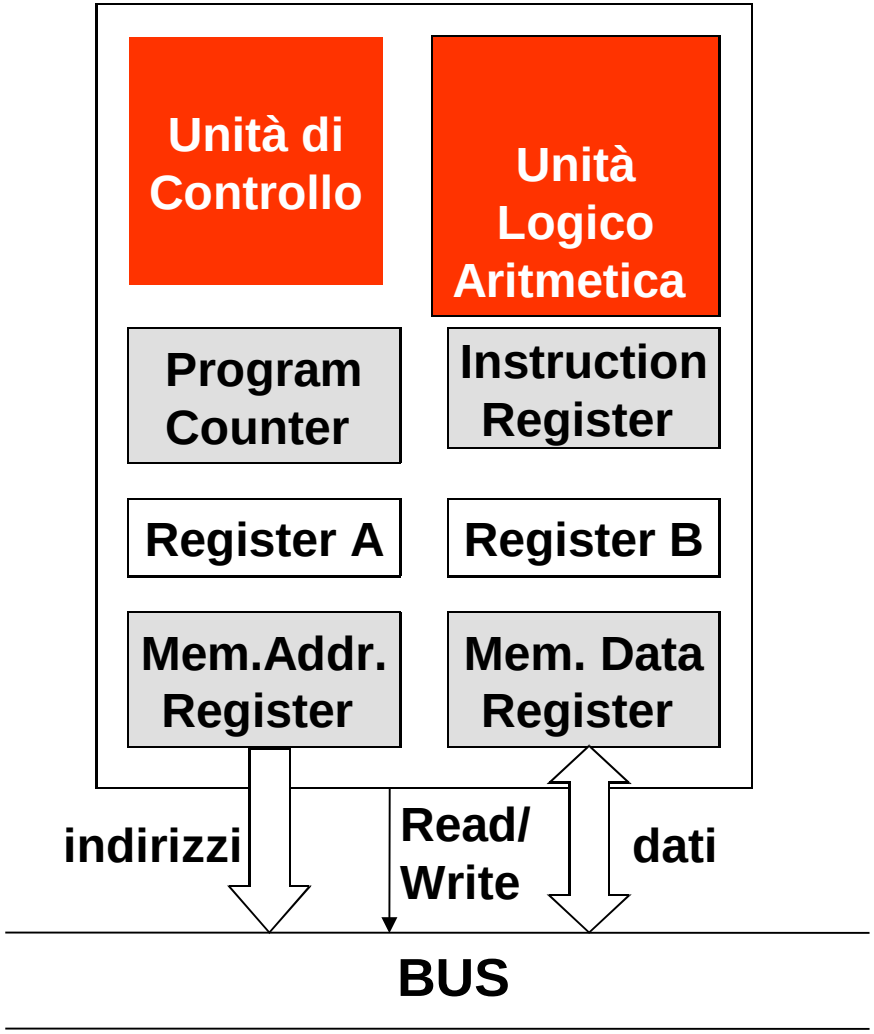
- **Registri generali**
- **Registri speciali**
  - Program Counter (PC)
  - Mem. Address Reg. (MAR)
  - Mem. Data Register (MDR)
  - Istrunction Register (IR)

I registri speciali non sono accessibili dalle istruzioni

Connessione della CPU con il sistema

I vari componenti interni della CPU sono comunicanti tramite connessioni interne.

La CPU è connessa al resto del sistema tramite il BUS (linee indirizzi, dati e controllo).



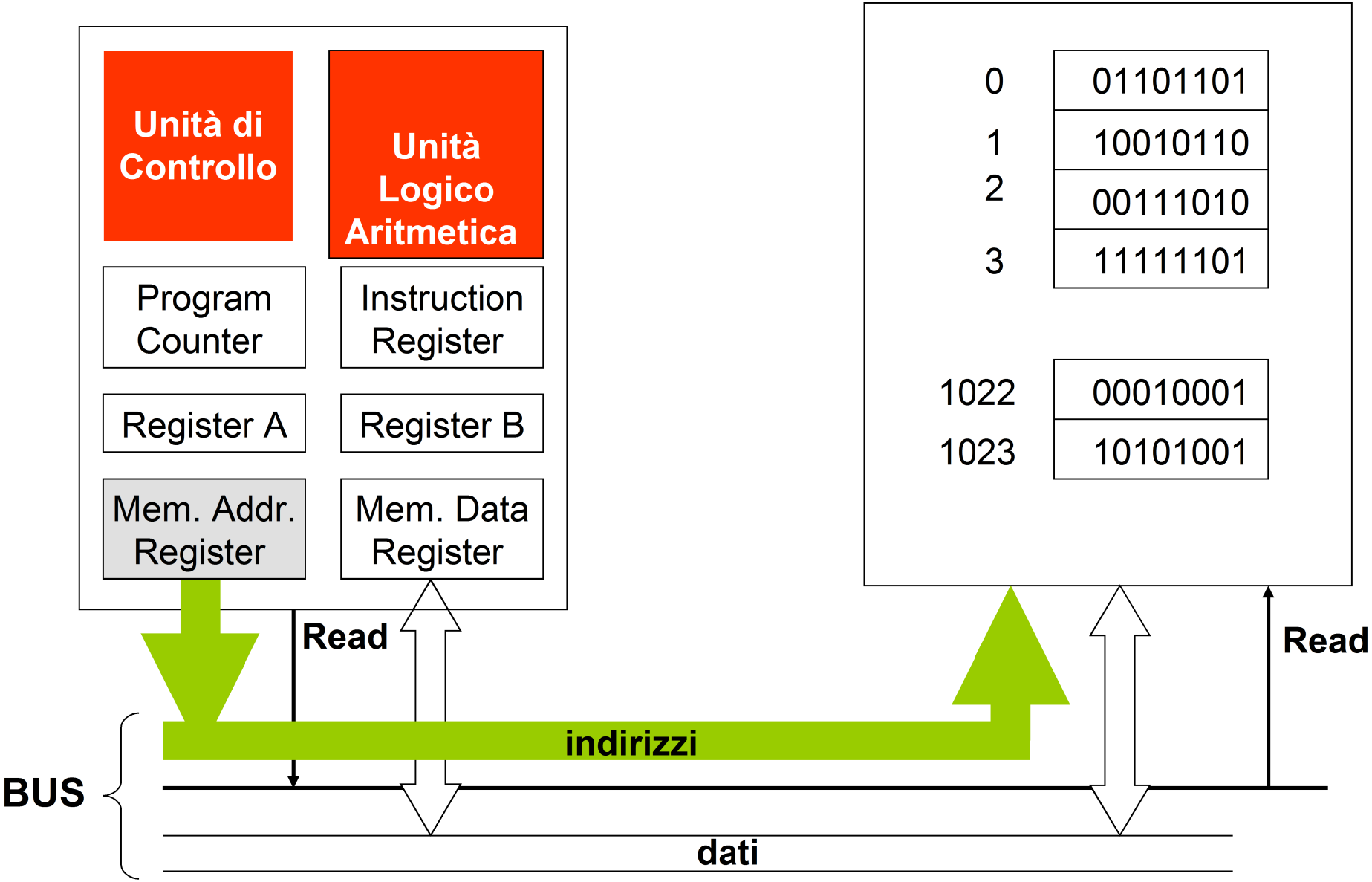
Trasferimento CPU-memoria

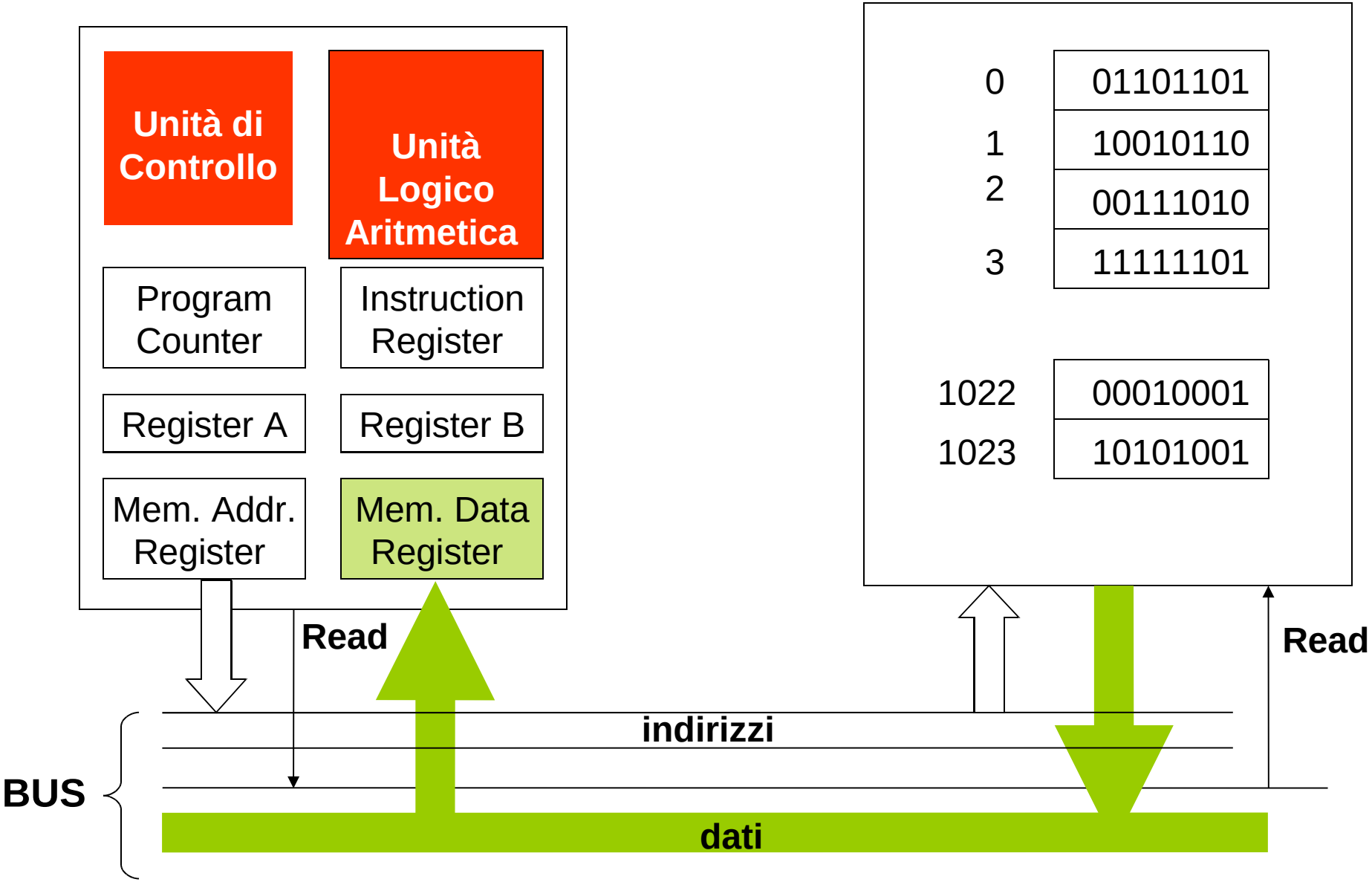
- Qualunque sia il trasferimento da realizzare, la CPU (master) deve precisare l'indirizzo del dato da trasferire.

- In queste operazioni, la memoria è comunque uno slave e "subisce" l'iniziativa della CPU, ricevendo da questa l'indirizzo del dato da trasferire e l'informazione sull'operazione da realizzare (lettura o scrittura)

Trasferimento memoria → CPU (lettura)

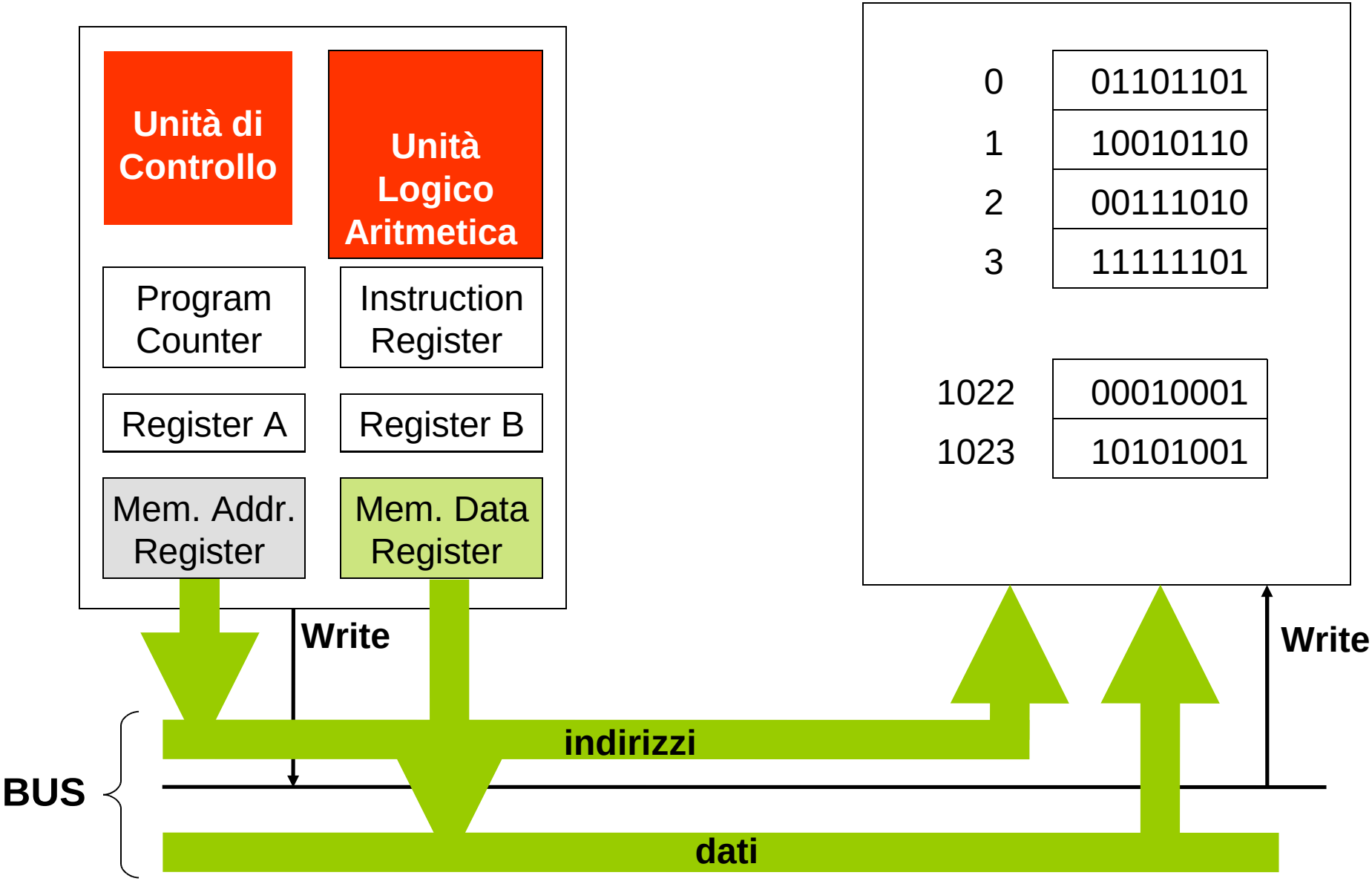
1. la CPU scrive l'indirizzo del dato da trasferire sul MAR che lo propagherà alle linee indirizzi del bus. Contemporaneamente, segnala sulle linee di controllo che si tratta di una lettura.
2. la memoria riceve, tramite il bus, l'indirizzo e l'indicazione dell'operazione da effettuare. Copia il dato dal registro individuato sulle linee dati del bus.
3. il dato richiesto, tramite le linee dati del bus, arriva al MDR della CPU. Da qui sarà spostato verso gli altri registi interni.

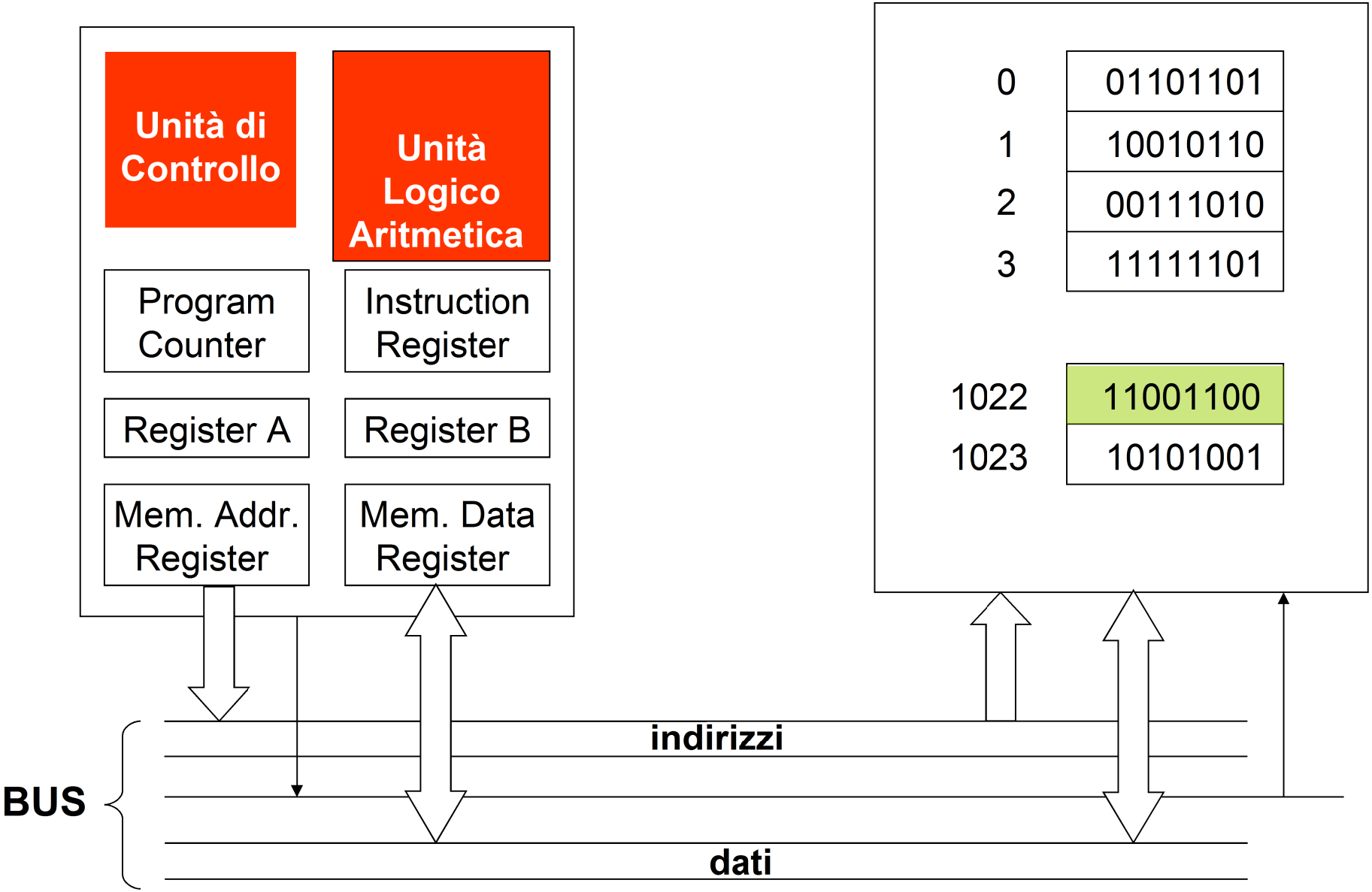

Unità di Controllo
Unità Logico Aritmetica
Program Counter
Instruction Register
Register A
Register B
Mem. Addr. Register
Mem. Data Register
0
01101101
1
10010110
2
00111010
3
11111101
1022
00010001
1023
10101001
Read
Read
indirizzi
BUS
dati


Unità di Controllo
Unità Logico Aritmetica
Program Counter
Instruction Register
Register A
Register B
Mem. Addr. Register
Mem. Data Register
0
01101101
1
10010110
2
00111010
3
11111101
1022
00010001
1023
10101001
Read
Read
indirizzi
BUS
dati

Trasferimento CPU → memoria (scrittura)

1. la CPU scrive l'indirizzo del dato da trasferire sul MAR, mentre il dato viene copiato sul MDR. Il contenuto dei due registri viene propagato sulle linee indirizzi e dati del bus. Contemporaneamente, la CPU segnala sulle linee di controllo che si tratta di una scrittura.
2. la memoria riceve, tramite il bus, l'indirizzo, il dato e l'indicazione dell'operazione da effettuare. Copia il dato dalle linee dati del bus al registro individuato dall'indirizzo.


Unità di Controllo
Unità Logico Aritmetica
Program Counter
Instruction Register
Register A
Register B
Mem. Addr. Register
Mem. Data Register
0
01101101
1
10010110
2
00111010
3
11111101
1022
00010001
1023
10101001
Write
Write
indirizzi
BUS
dati


Unità di Controllo
Unità Logico Aritmetica
Program Counter
Instruction Register
Register A
Register B
Mem. Addr. Register
Mem. Data Register
0
01101101
1
10010110
2
00111010
3
11111101
1022
11001100
1023
10101001
indirizzi
BUS
dati

Un calcolatore basato sul modello di von Neumann permette l'esecuzione di un *programma*, cioè di una sequenza di istruzioni descritte nel linguaggio interpretabile dal calcolatore che realizzano un particolare algoritmo, ma quali sono le caratteristiche di tale linguaggio ?

- è codificato tramite sequenze di bit
- accede ai dati tramite gli indirizzi di memoria o i registri interni della CPU
- ogni istruzione può compiere solo azioni molto semplici
- non gestisce direttamente i tipi di dati di interesse
- è strettamente legato alla particolare macchina su cui è definito

Non a caso viene definito *linguaggio macchina*

Se si volesse implementare un dato algoritmo attraverso la scrittura di un programma sarebbe quindi necessario:

- conoscere dettagliatamente tutti i codici operativi e la loro codifica
- decidere in quali registri (di memoria o interni alla CPU) vadano memorizzati i dati
- determinare, per ogni singola operazione richiesta dall'algoritmo, la sequenza di istruzioni in linguaggio macchina che la realizzano
- definire un'opportuna tecnica di codifica per ogni tipo di dati considerato
- limitarsi a utilizzare solo i calcolatori per cui esista una tale competenza, tenendo comunque presente che il programma scritto per un certo calcolatore non è eseguibile su altre macchine

Impresa difficile, ma non impossibile

Linguaggio di programmazione

- linguaggio formale, con costrutti precisi per la definizione dei dati e delle operazioni
- gestione completa dei tipi fondamentali; possibilità di definire tipi strutturati
- costrutti che realizzano le principali azioni elaborative richieste

Calcolatore

- linguaggio rigido e complicato
- gestione dei tipi quasi nulla
- istruzioni estremamente semplici

Linguaggio di programmazione

- linguaggio formale, con costrutti precisi per la definizione dei dati e delle operazioni
- gestione completa dei tipi fondamentali; possibilità di definire tipi strutturati
- costrutti che realizzano le principali azioni elaborative richieste

orientato al linguaggio (*front end*)

Compilatore

Fondamenti di Informatica

orientato alla macchina (*back end*)

Calcolatore

- linguaggio rigido e complicato
- gestione dei tipi quasi nulla
- istruzioni estremamente semplici

Calcolatori Elettronici

Perchè studiare l'architettura del Calcolatore ?

- Comprendere quali aspetti del SW e dell'HW hanno un impatto sulle prestazioni dei programmi
- Comprendere come un programmatore può migliorare le prestazioni del suo programma
- Comprendere quali caratteristiche bisogna considerare per valutare un sistema di elaborazione in uno specifico contesto applicativo

Quale processore ? MIPS

- **MIPS**: un'azienda che ha costruito una delle prime architetture RISC commerciali
- Studieremo l'architettura MIPS in qualche dettaglio
- Perchè MIPS invece di (es.) Intel 80x86 ?
  - L'architettura e l'ISA del MIPS sono molto più semplici ed eleganti
  - Il MIPS è largamente utilizzato in applicazioni "embedded", contrariamente all'INTEL 80x86 che è praticamente limitato al solo segmento del personal computer

Architettura del processore MIPS

Microprocessor without Interlocking Pipe Stages

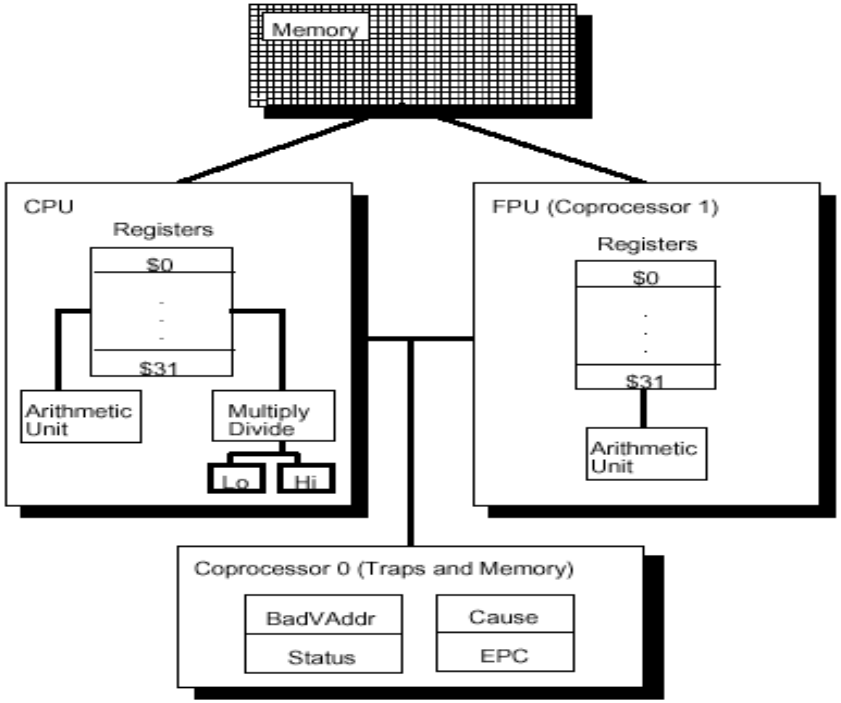
- Architettura Load/Store con istruzioni aritmetiche registro-registro a 3 operandi
- Istruzioni di 32-bit - 3 Formati (R, I, J)
- 32 registri generali di 32 bit (R0 contiene 0, R31 riceve l'indirizzo di ritorno) (+ HI, LO)
- Modi d'indirizzamento: Register, Immediate, Base+Offset, PC-relative
- Immediati a 16-bit + istruzione LUI

Architettura del processore MIPS

- Supporto per interi in complemento a 2 di 8 (byte), 16 (halfword) e 32 (word) bit e, con coprocesore opzionale, per numeri floating-point IEEE 754 singola e doppia precisione
- Branch semplici senza codici di condizione
- *Delayed branch* (l'istruzione dopo il salto viene comunque eseguita) e *Delayed load* (l'istruzione dopo una load non deve usare il registro caricato), senza interlock

Coprocessori

- Può supportare fino a 4 coprocessori, numerati da 0 a 3
- Il coprocessore di controllo del sistema (coprocessore 0) è integrato nel chip e gestisce la memoria e le eccezioni
- Il coprocessore floating-point (coprocessore 1) opzionale ha 32 registri di 32-bit ($f0 - $f31), di cui sono utilizzabili quelli di posto pari in semplice o doppia precisione



Registri del MIPS e convenzione di uso

32 registri generali da 32 bit

Register name	Number	Usage
$zero	0	constant 0
$at	1	reserved for assembler
$v0	2	expression evaluation and results of a function
$v1	3	expression evaluation and results of a function
$a0	4	argument 1
$a1	5	argument 2
$a2	6	argument 3
$a3	7	argument 4
$t0	8	temporary (not preserved across call)
$t1	9	temporary (not preserved across call)
$t2	10	temporary (not preserved across call)
$t3	11	temporary (not preserved across call)
$t4	12	temporary (not preserved across call)
$t5	13	temporary (not preserved across call)
$t6	14	temporary (not preserved across call)
$t7	15	temporary (not preserved across call)
$s0	16	saved temporary (preserved across call)
$s1	17	saved temporary (preserved across call)
$s2	18	saved temporary (preserved across call)
$s3	19	saved temporary (preserved across call)
$s4	20	saved temporary (preserved across call)
$s5	21	saved temporary (preserved across call)
$s6	22	saved temporary (preserved across call)
$s7	23	saved temporary (preserved across call)
$t8	24	temporary (not preserved across call)
$t9	25	temporary (not preserved across call)
$k0	26	reserved for OS kernel
$k1	27	reserved for OS kernel
$gp	28	pointer to global area
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address (used by function call)

registri speciali

PC

HI
LO

Gestione degli indirizzi di memoria

- Spazio di indirizzi di 2^{32} byte (4 Gigabyte, con i 2 superiori riservati al S.O.), ossia 2^{30} word
- L'indirizzamento è al byte (incremento di 4 per passare da una word alla successiva)
- L'indirizzo di una word è quello del suo primo byte (byte di indirizzo minore)
- Negli accessi, l'indirizzo di un dato di s byte deve essere allineato, ossia A mod s = 0 (esistono istruzioni per accedere a dati disallineati)
- L'ordinamento dei byte in una word può essere sia *big-endian* (il primo byte è quello più significativo) che *little-endian* (il primo byte è quello meno significativo), in dipendenza del valore logico su di un pin

D:\2010F CSC 320\MIPS assembly code\FibonnaciWithStack.asm* - MARS 4.0.1

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit | Execute

FibonnaciWithStack.asm*

```
1 # Ken Vollmar
2 # Feb. 13, 2002
3 # Computing the i-th Fibonacci number using the stack
4
5
6 .data            # Put any data initializations between .data and .text (before or after __start)
7
8 str1:   .asciiz "Which Fibonacci number do you want? "
9 str2:   .asciiz "The Fibonacci number is "
10
11.text
12                add
13
14                addi                                        call for print string
15                lui                                         expected at 0x10010000
16                                                            bits to zero.
17                sysc
18
19                addi                                        call for read integer
20                sysc
21
22
23                add $s0, $zero, $zero  # li     $s0, 0          # s0 will hold the i-th Fibonacci number
24                add $s1, $zero, $zero  # li     $s1, 0            # s1 is the count of items on the stack
25
26                addi $sp, $sp, -4       # adjust stack prior to push (add -4)
27                sw   $v0, 0($sp)        # Push number onto stack (the i-th Fib. number desired)
28                addi $s1, $s1, 1        # count of number of items on stack
29
30loop:   lw $t0, 0($sp)          # Pop one item off stack
31                addi $sp, $sp, 4        # adjust stack pointer after pop (add +4)
32                addi $s1, $s1, -1       # count of number of items on stack
33
34                addi $t7, $t0, -1       # $t7 = (i-1)
35                bgtz $t7, notbase       # if ( (i-1) > 0) then put other elements on stack because F_i is not yet base case
36base:   addi $s0, $s0, 1        # This is Fib. base case, either F_1 = 1 or F_0 = 1
37                                                # Add one to the accumulating Fibonnaci number contained in $s0
38                j loopbot
39
40notbase:
41                addi $t0, $t0, -1       # This gives the value of (i-1) so we can find F_(i-1)
42                addi $sp, $sp, -4       # adjust stack prior to push (add -4)
```

add	Addition with overflow
add.d	Floating point addition double precision
add.s	Floating point addition single precision
addi	Addition immediate with overflow
addiu	Addition immediate unsigned without overflow
addu	Addition unsigned without overflow

Line: 12 Column: 6 Show Line Numbers

Mars Messages | Run I/O

Clear

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194304
hi		0
lo		0