Introduzione al linguaggio MIPS

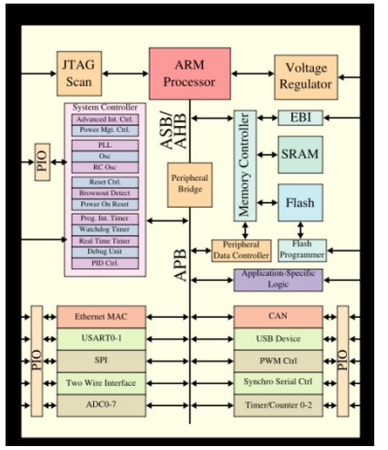
M. Rebaudengo, M. Sonza Reorda, L. Sterpone

Politecnico di Torino
Dip. di Automatica e Informatica

I processori: cosa sono

Il processore è il modulo principale di un sistema a processore.

SoC



Un processore è un modulo che può corrispondere

- **a una parte (denominata *core*) di un circuito integrato (SoC o microcontrollore)**
- **a un circuito integrato a se stante.**

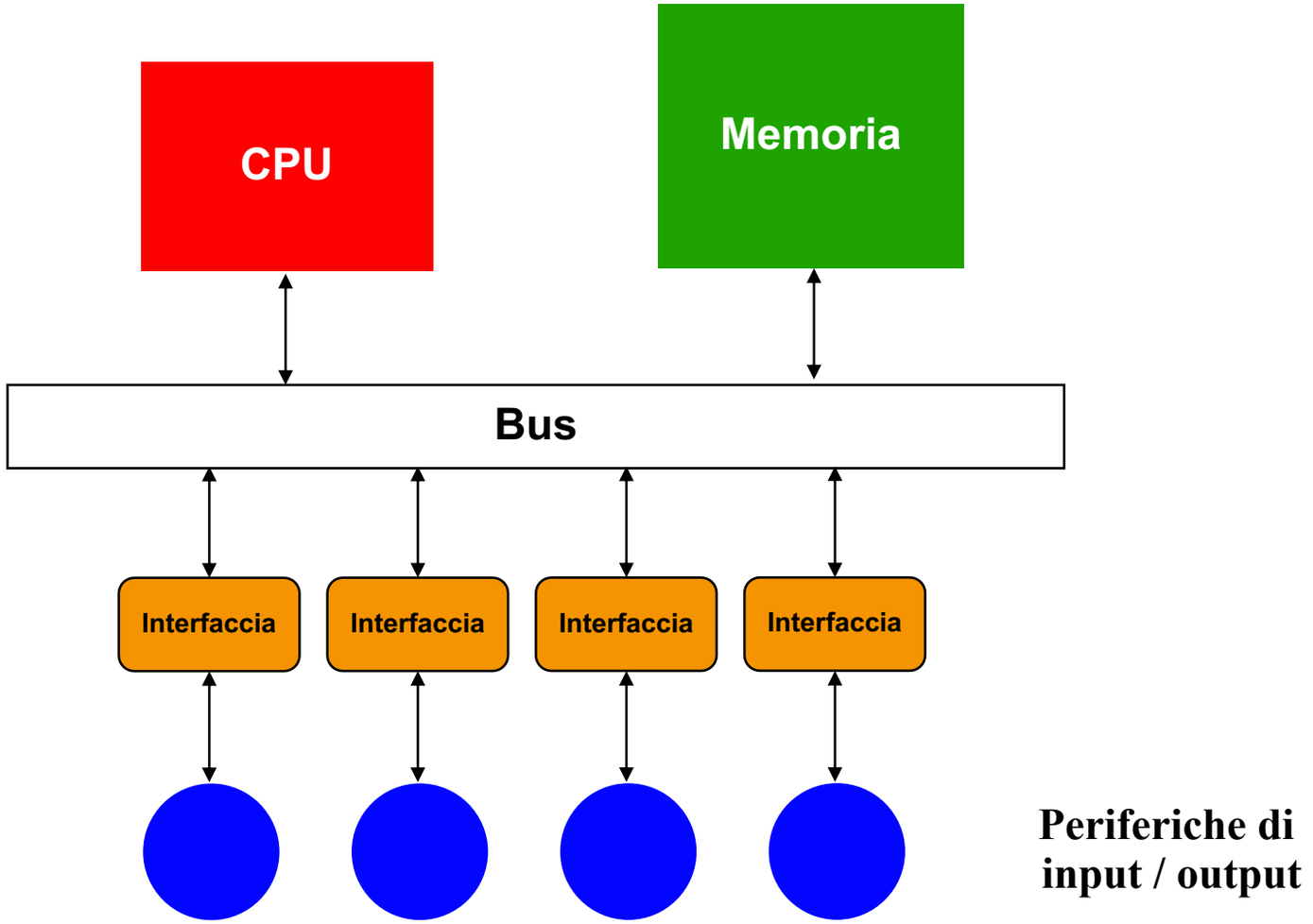
Esistono numerose tipologie di processori in termini di complessità, potenza di calcolo, architettura, ecc.

I processori: cosa fanno

Un processore è un dispositivo che compie 2 tipi di operazioni:

- **esegue istruzioni (il cui codice sta in memoria)**
- **interagisce con il mondo esterno (attraverso opportune interfacce).**

Sistema a processore



Periferiche di
input / output

Istruzioni

L'esecuzione di ciascuna istruzione si compone di 2 fasi:

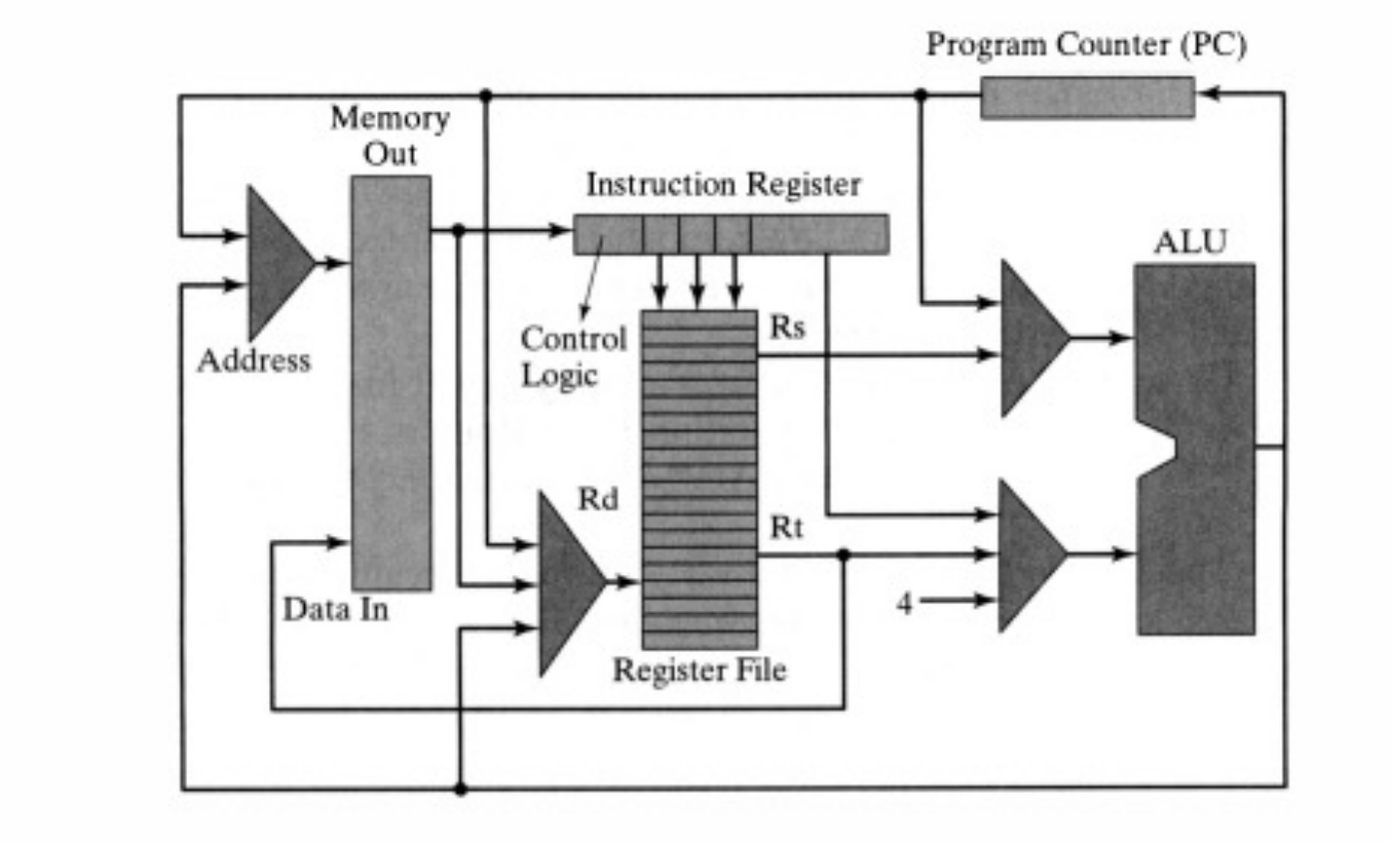
- ***fetch*: il codice dell'istruzione viene letto dalla memoria**
- ***execute*: il codice viene prima decodificato, poi eseguito. Questo comporta normalmente l'accesso ad uno o più operandi, l'esecuzione di una operazione su di essi, la scrittura del risultato.**

La combinazione delle due fasi si dice *ciclo di istruzione*.

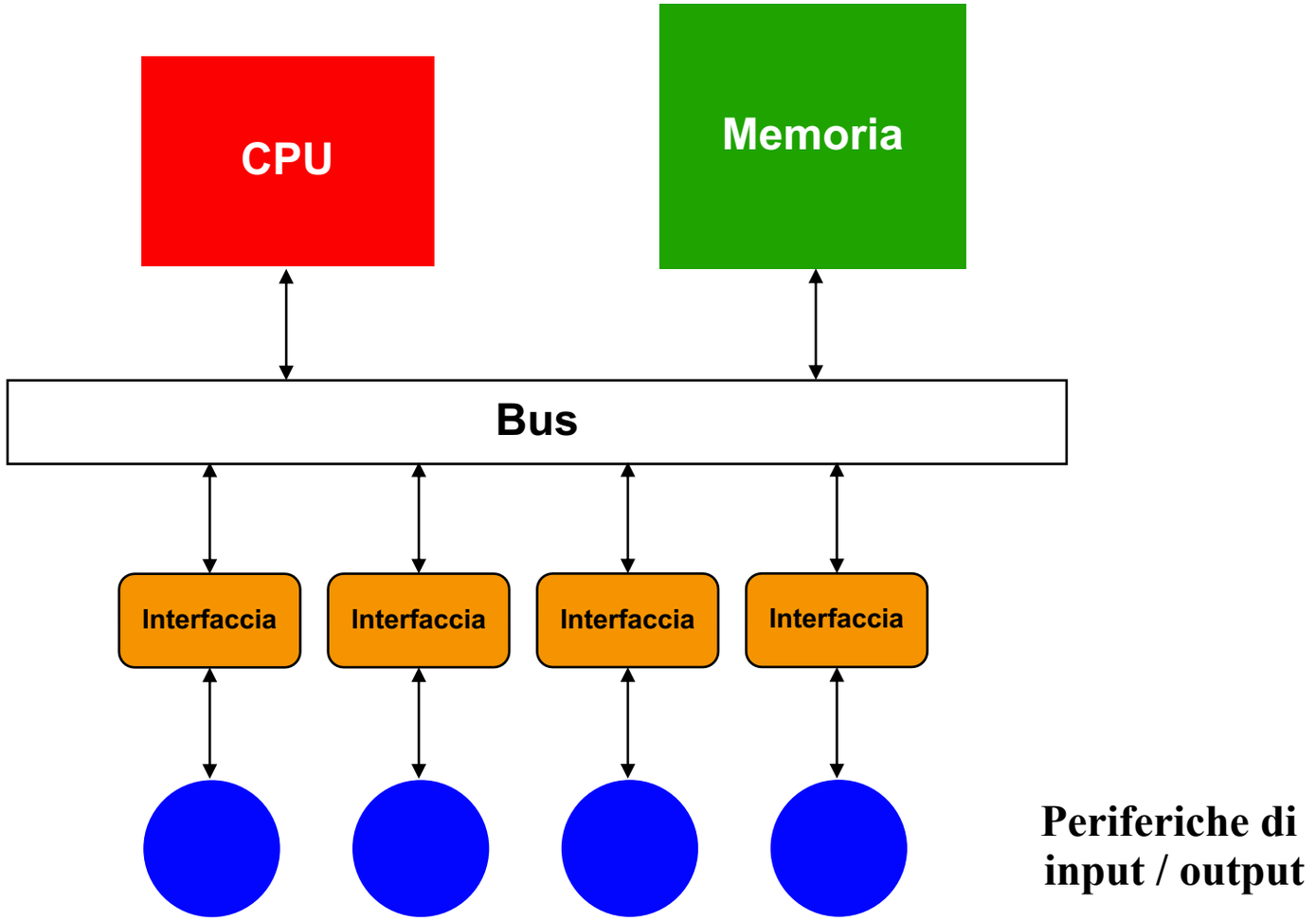
Registri

- **Il tempo per accedere alla memoria è normalmente superiore al tempo necessario alla CPU per processare i dati**
- **L'accesso alla memoria rappresenta quindi un *collo di bottiglia* (in inglese *bottleneck*) per le prestazioni delle CPU**
- **Per questa ragione all'interno della CPU sono presenti alcune celle di memoria particolarmente veloci, note come *registri***
- **Ove possibile le operazioni vengono svolte utilizzando i registri per contenere operandi e risultato.**

Modello architetturale della CPU MIPS

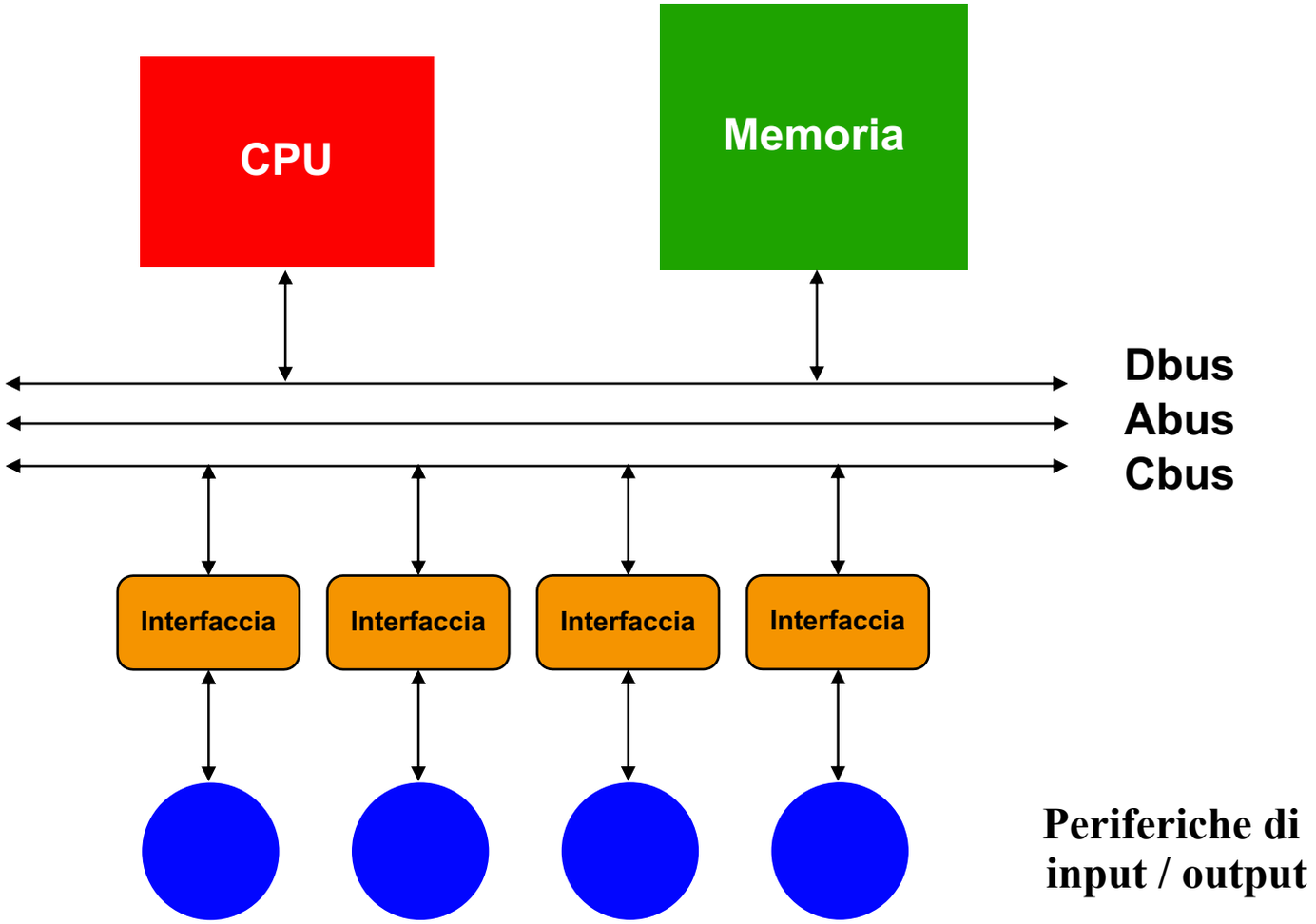


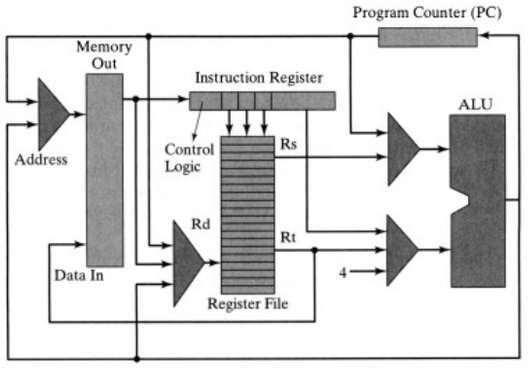
Sistema a processore



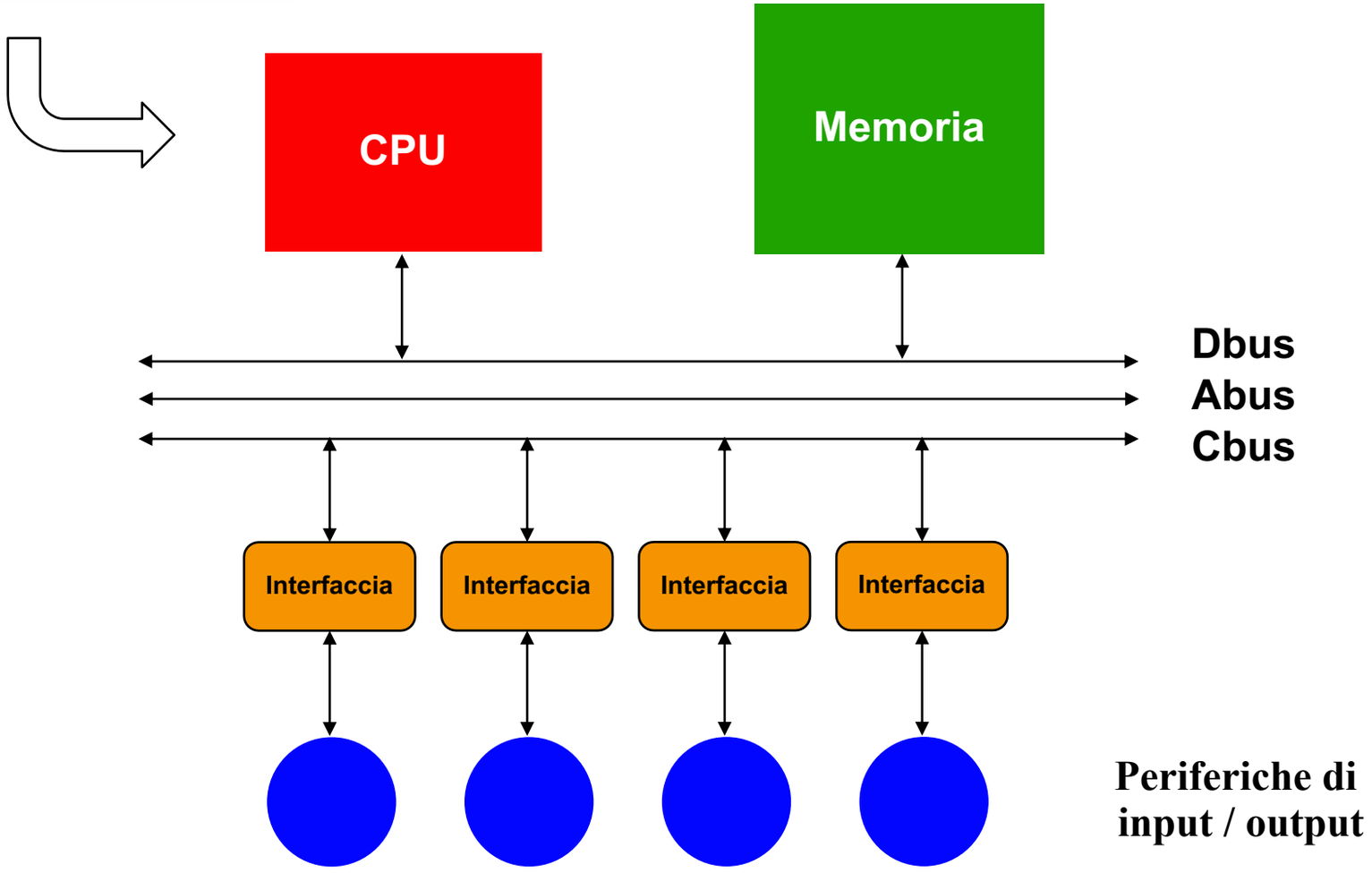
Periferiche di input / output

Sistema MIPS

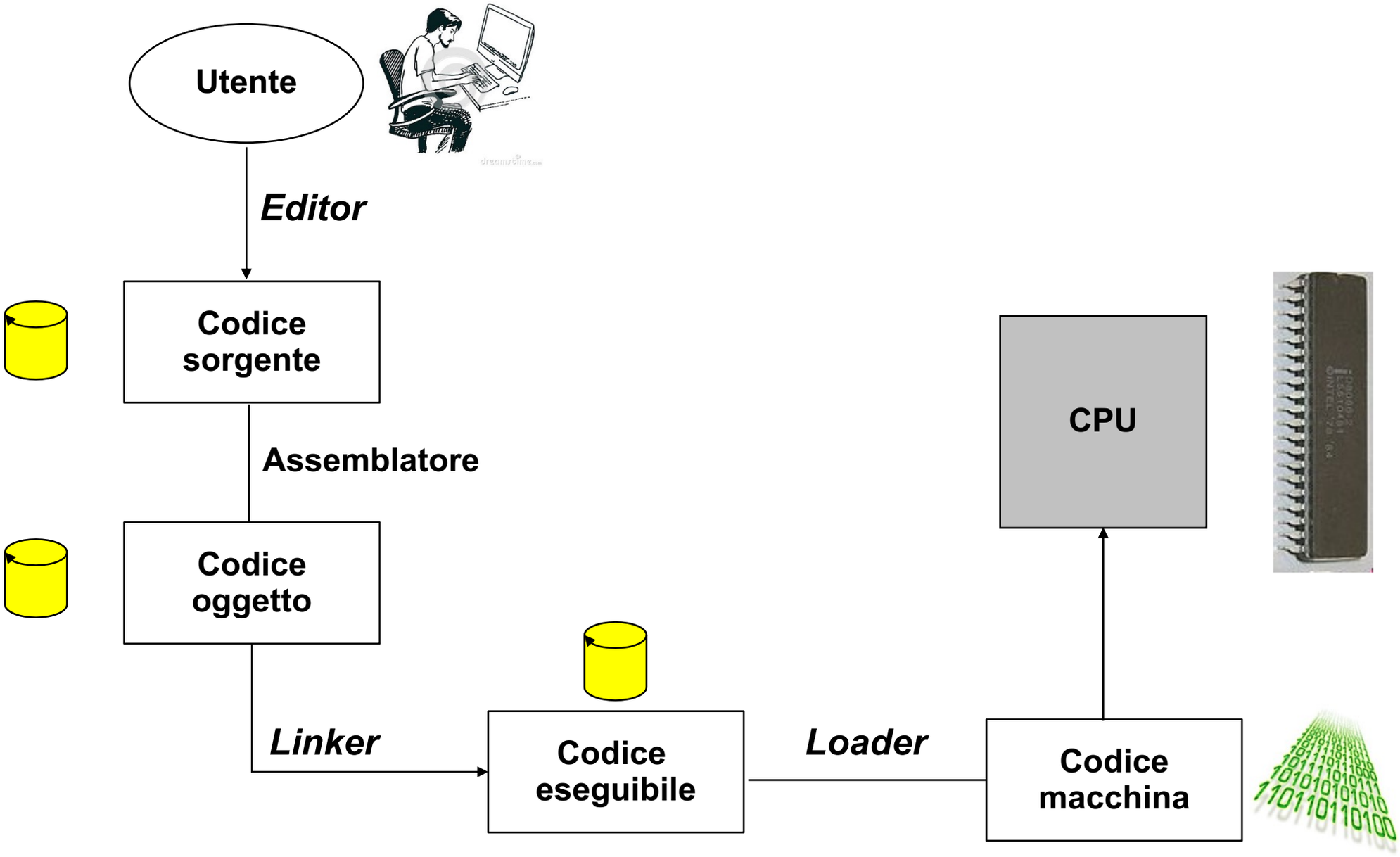




Sistema MIPS



Ciclo di vita di un programma



Assembly

- **Rappresentazione simbolica del linguaggio macchina**
- **Vero e proprio linguaggio di programmazione**
- **Più comprensibile del linguaggio macchina in quanto utilizza simboli invece che sequenze di bit**
- **Rispetto ai linguaggi ad alto livello...**
  - **Assembly è il linguaggio target di compilazione di codice in linguaggi ad alto livello**
  - **Assembly fornisce limitate forme di controllo del flusso**
  - **Assembly non prevede articolate strutture dati**

Assembly

- **VANTAGGI:**
  - **Visibilità diretta dell'hardware**
  - **Massimo sfruttamento delle potenzialità HW della macchina**
  - **Ottimizzazione delle prestazioni**
- **SVANTAGGI:**
  - **Mancanza di portabilità dei programmi**
  - **Maggior costo di sviluppo**
  - **Maggiore lunghezza e difficoltà di comprensione**

C and Assembly

C:

```
main()
  { int i;
    int sum = 0;
    for (i=0; i<=100; i=i+1)
      sum = sum + i*i;
  }
```

Assembly-MIPS:

```
 main:
    subu $sp, $sp, 32
    sw $ra, 20($sp)
    sw $a0, 32 ($sp)
    sw $0, 24($sp)
    sw $0, 28($sp)
loop:
    lw $t6, 28($sp)
    lw $t8, 24($sp)
    mult $t4, $t6, $t6
    addu $t9, $t8, $t4
    addu $t9, $t8, $t7
    sw $t9, 24($sp)
    addu $t7, $t6, 1
    sw $t7, 28($sp)
    bne $t5, 100, loop
```

Azzeramento di un registro

```
int a;

a = 0;
```

```
#  write 0 in $t0


.text
.globl main
.ent main
main:
    and   $t0, $0, $0


    li      $v0, 10
    syscall
.end main
```

Somma di due valori

```
int     opd1,
        opd2,
        result;
```

```
#  reading 2 values from memory, adding and writing them back
.data
opd1:       .word 0x00005678
opd2:       .word 0x12340000
result:     .space 4

.text
.globl main
.ent main
main:
            la      $t0, opd1
            lw      $t1, ($t0)
            la      $t0, opd2
            lw      $t2, ($t0)
            add     $t3, $t1, $t2

            la      $t0, result
            sw      $t3, ($t0)
            li      $v0, 10
            syscall
.end main
```

```
result = opd1 + opd2;
```

Somma degli elementi di un vettore (I)

```
int    vett[5],
       result,
       tmp=0;
```

```
tmp = 0;
tmp += vett[0];
tmp += vett[1];
tmp += vett[2];
tmp += vett[3];
tmp += vett[4];
result = tmp;
```

```
#  adds all the elements of a vector
.data
vett:           .word 5, 7, 3, 4, 3
result:         .word 0

.text
.globl main
.ent main
main:
        la      $t3, vett       #load address of vett
        and     $t4, $0, $0     # temporary register = 0
        lw      $t0, ($t3)      # load first element from vett
        add     $t4, $t4, $t0   # add to the temp register
        addi    $t3, $t3, 4     # update index
        lw      $t0, ($t3)      # load first element from vett
        add     $t4, $t4, $t0   # add to the temp register
        addi    $t3, $t3, 4     # update index
        lw      $t0, ($t3)      # load first element from vett
        add     $t4, $t4, $t0   # add to the temp register
        addi    $t3, $t3, 4     # update index
        lw      $t0, ($t3)      # load first element from vett
        add     $t4, $t4, $t0   # add to the temp register
        addi    $t3, $t3, 4     # update index
        lw      $t0, ($t3)      # load first element from vett
        add     $t4, $t4, $t0   # add to the temp register

        la      $t3, result
        sw      $t4, ($t3)      # write result to memory

        li      $v0, 10
        syscall
.end main
```

Somma degli elementi di un vettore (II)

```
int     vett[5],
        result,
        tmp=0,
        i;


for(i=0; i<5; i++)
        tmp += vett[i];

result = tmp;
```

```
#  adds all the elements of a vector
.data
vett:     .word 5, 7, 3, 4, 3
result:   .word 0

.text
.globl main
.ent main
main:
          la      $t1, vett
          and     $t3, $0, $0       # index
          and     $t4, $0, $0       # temporary register
lab1:     lw      $t0, ($t1)        # load first element from vett
          add     $t4, $t4, $t0     # add to the temp register
          addi    $t1, $t1, 4
          addi    $t3, $t3, 1       # update index
          bne     $t3, 5, lab1      # repeat 5 times

          la      $t1, result
          sw      $t4, ($t1)

          li      $v0, 10

          syscall
.end main
```

Lettura e visualizzazione di un vettore di caratteri

```
# legge DIM elementi di un vettore e li visualizza sulla console (schermo) in ordine inverso
.data
DIM = 5
vett:       .space DIM

.text
.globl main
.ent main
main:
            la      $t2, vett
            and     $t3, $0, $0             # index

lab1:       li      $v0, 12                 # read one character
            syscall

            sb      $v0, ($t2)
            addi    $t2, $t2, 1
            addi    $t3, $t3, 1             # update index
            bne     $t3, DIM, lab1                  # repeat 5 times

lab2:       addi    $t3, $t3, -1            # update index
            addi    $t2, $t2, -1

            lb      $a0, ($t2)
            li      $v0, 11                 # write one character
            syscall


            bne     $t3, 0, lab2            # repeat 5 times

            li      $v0, 10
            syscall
.end main
```

```
char    vett[5];
int i;

for(i=0; i<5; i++)
        scanf("%c", &vett[i]);

for(i=4; i>=0; i--)
        printf("%c ", vett[i]);
```