Introduzione al linguaggio Assembler 8086

M. Rebaudengo **M. Sonza Reorda**

Politecnico di Torino
Dip. di Automatica e Informatica

I processori: cosa sono

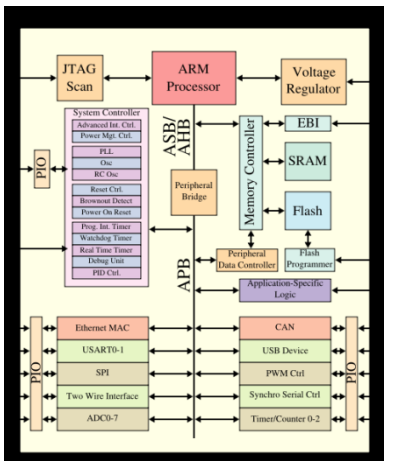
Il processore è il modulo principale di un sistema a processore.

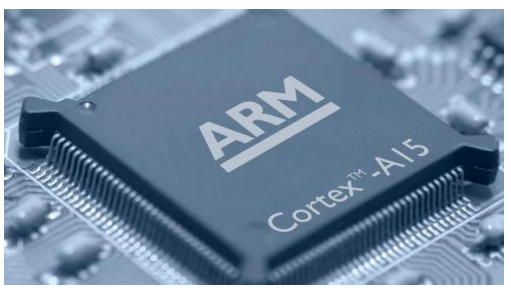
Un processore è un modulo che può corrispondere

- **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.

SoC



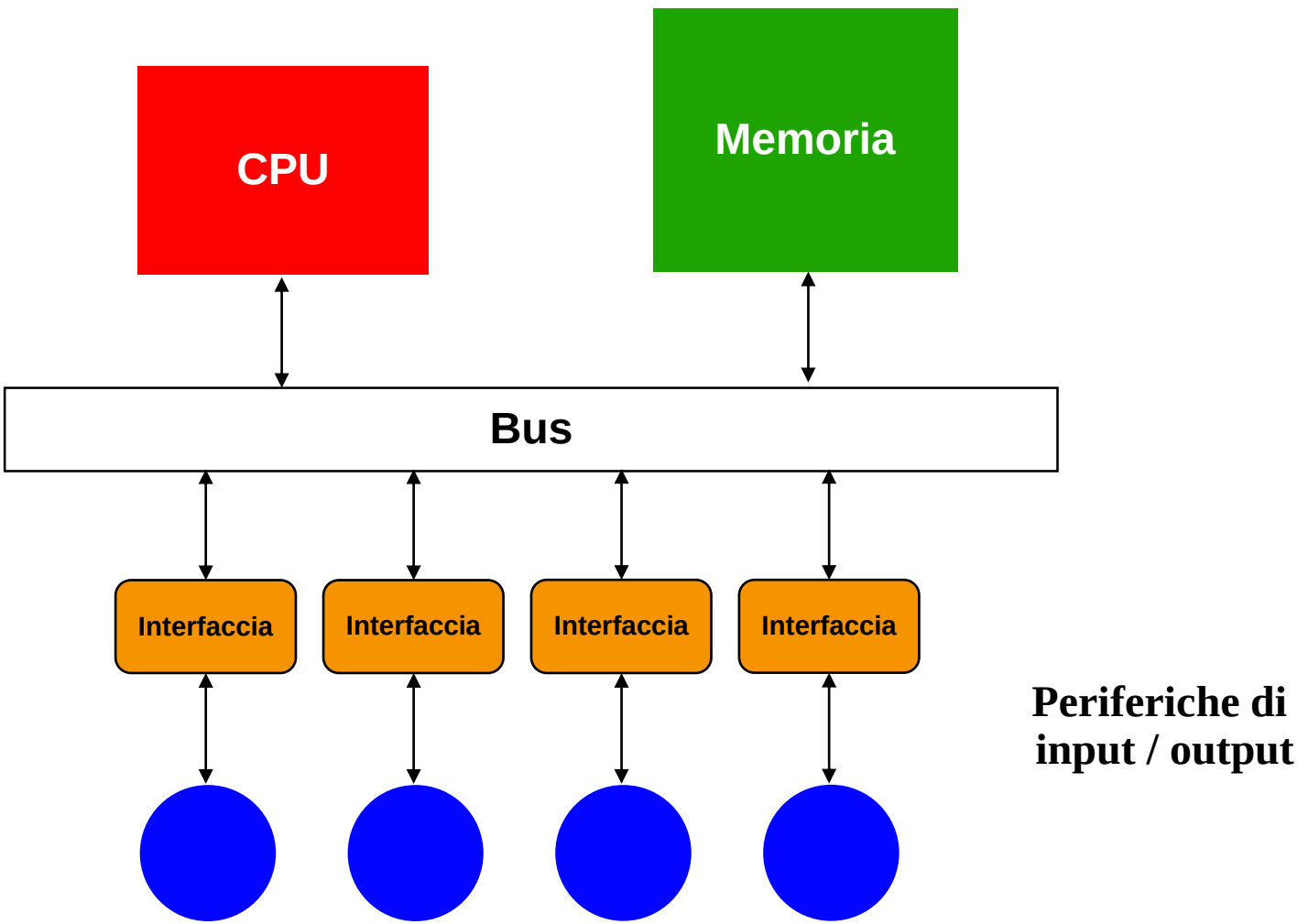


I processori: cosa fanno

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



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* 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.**

CPU elementare

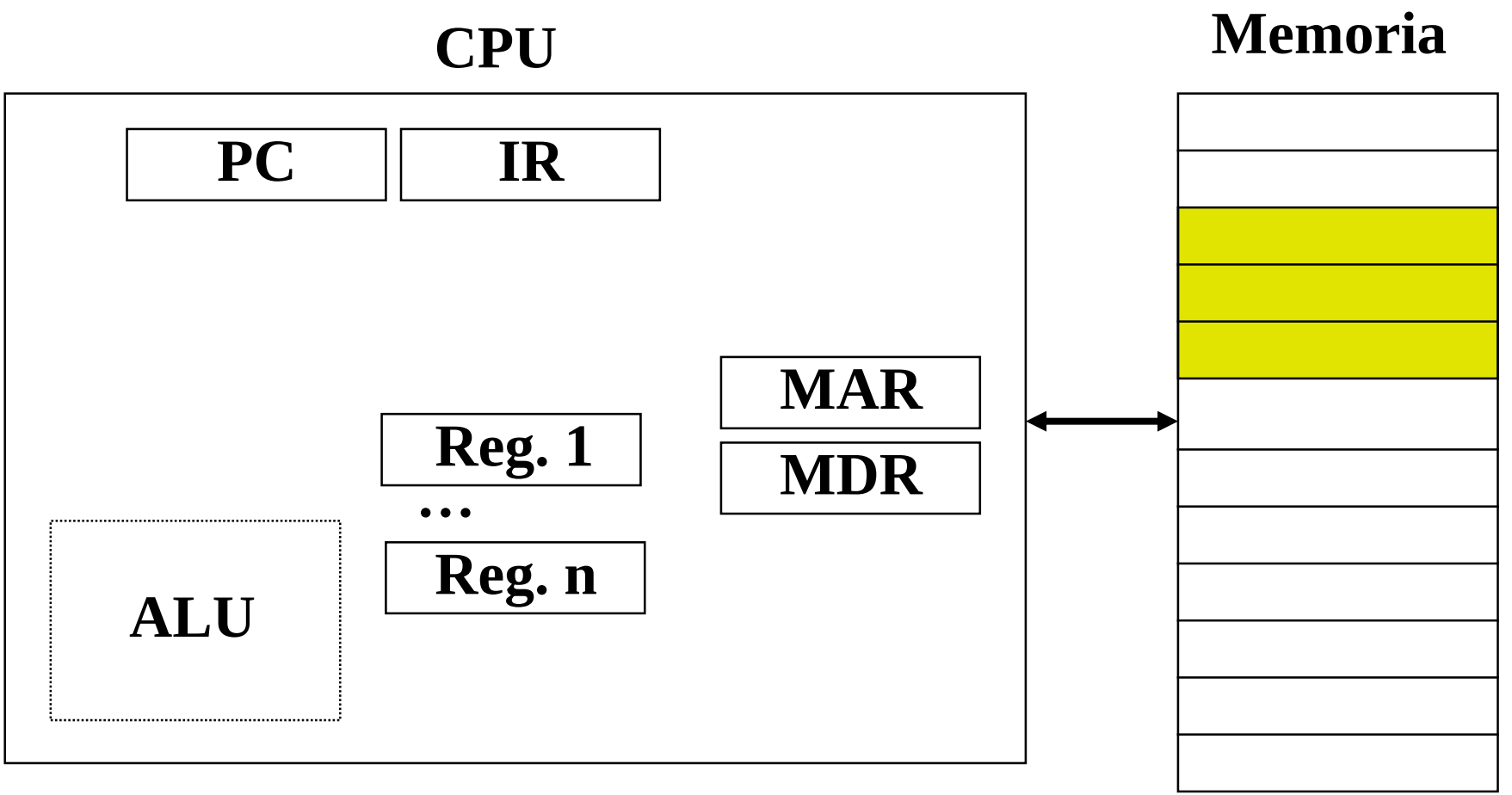
Alcuni componenti fondamentali di una CPU

- **`Program Counter` (PC)**
  - **memorizza l'indirizzo della prossima istruzione**
- **`Instruction Register` (IR)**
  - **Memorizza l'istruzione in corso di esecuzione**
- **`Registers` (Reg. 1..n)**
  - **Memorizzano le variabili ed i risultati temporanei**
- **`Arithmetic and Logic Unit` (ALU)**
  - **Esegue le operazioni aritmetiche e logiche**

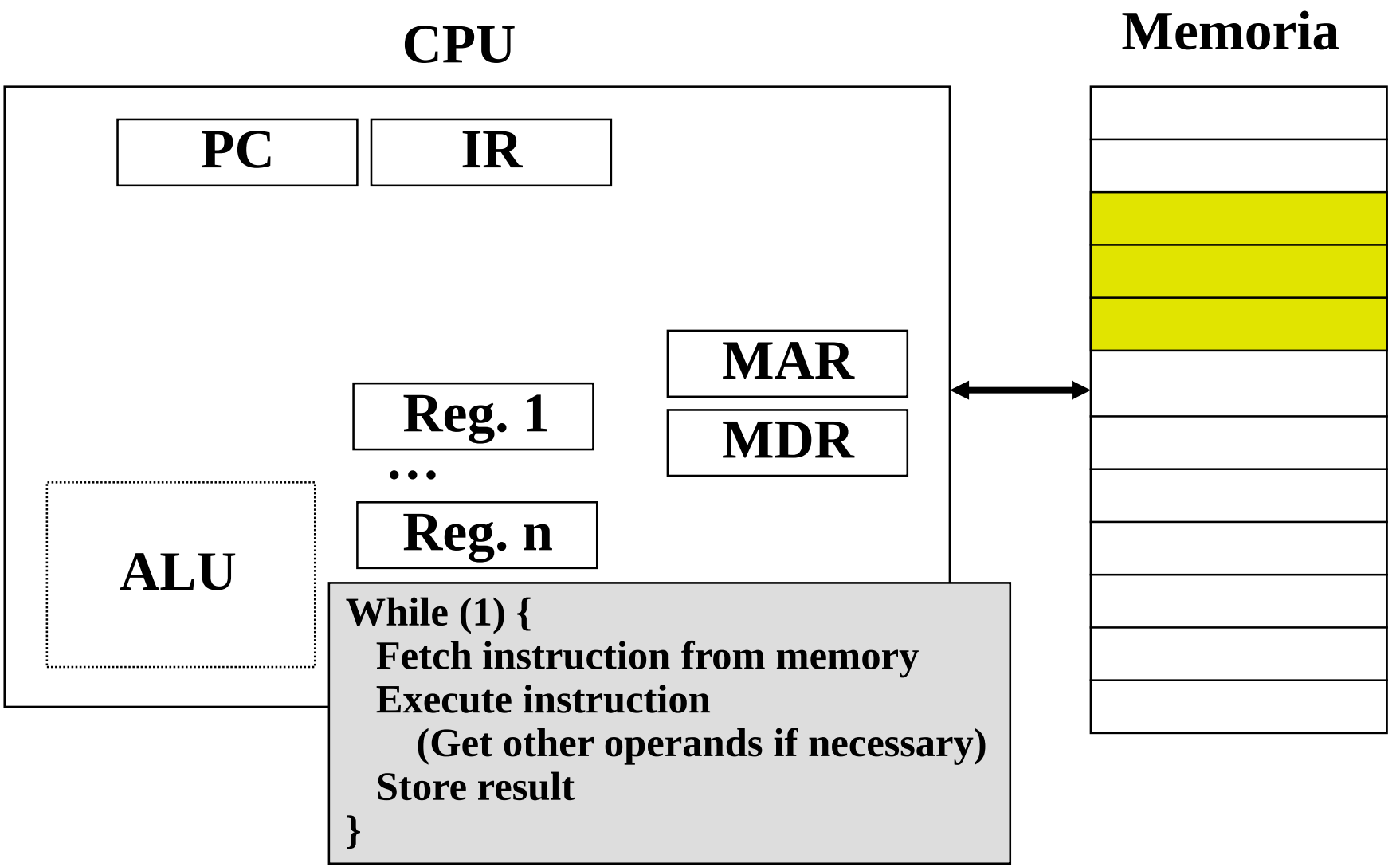
CPU elementare

- **`Memory Address Register` (MAR)**
  - **Contiene l'indirizzo della locazione di memoria da leggere/scrivere**
- **`Memory Data Register` (MDR)**
  - **Contiene il dato da scrivere/leggere in memoria**
- **`Stack Pointer` (SP)**
  - **Contiene l'indirizzo dell'ultima locazione memorizzata nello stack**
- **`Processor Status Word` (PSW)**
  - **Contiene i bit di flag e di controllo**

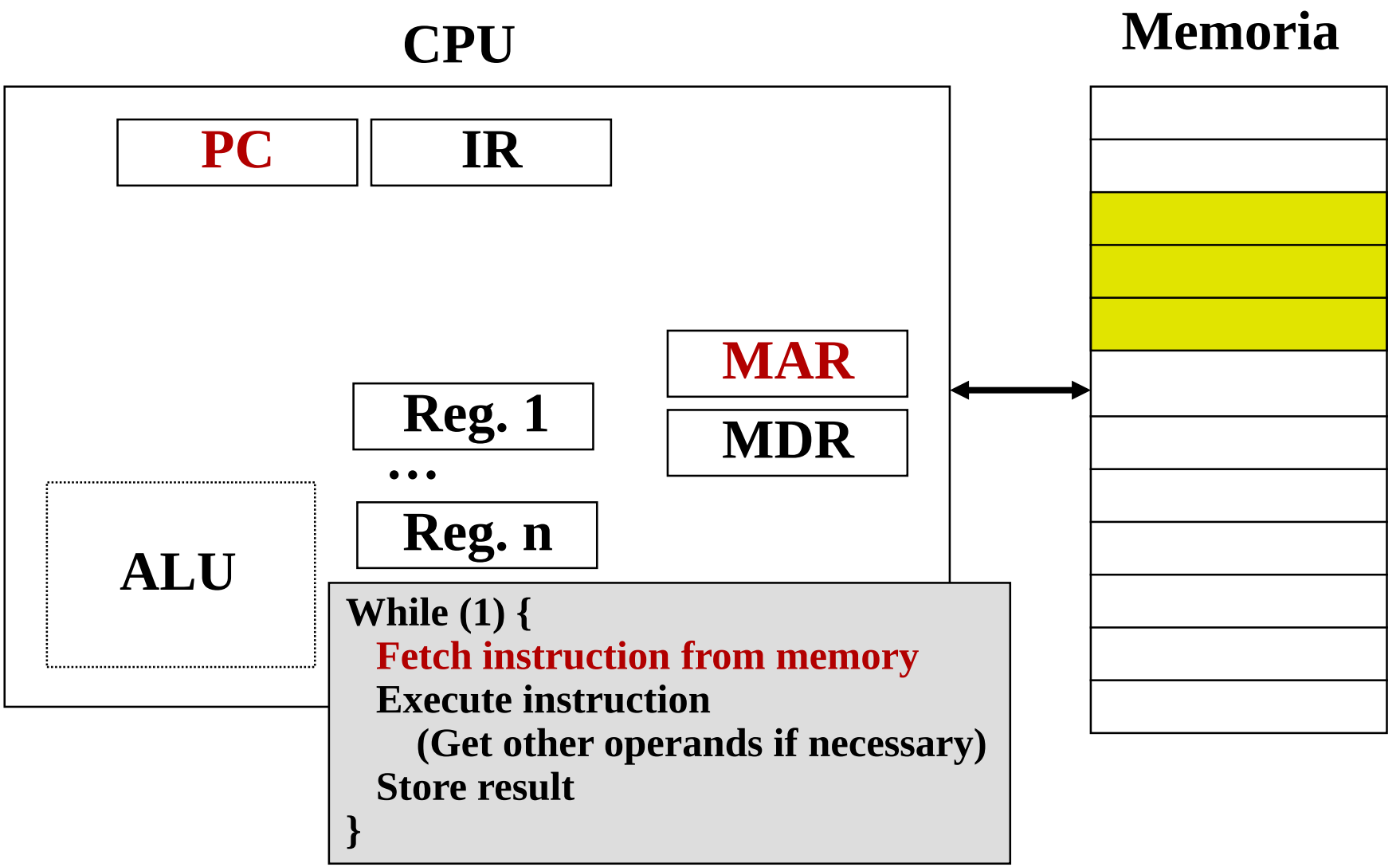
Fetch/decode/execute cycle



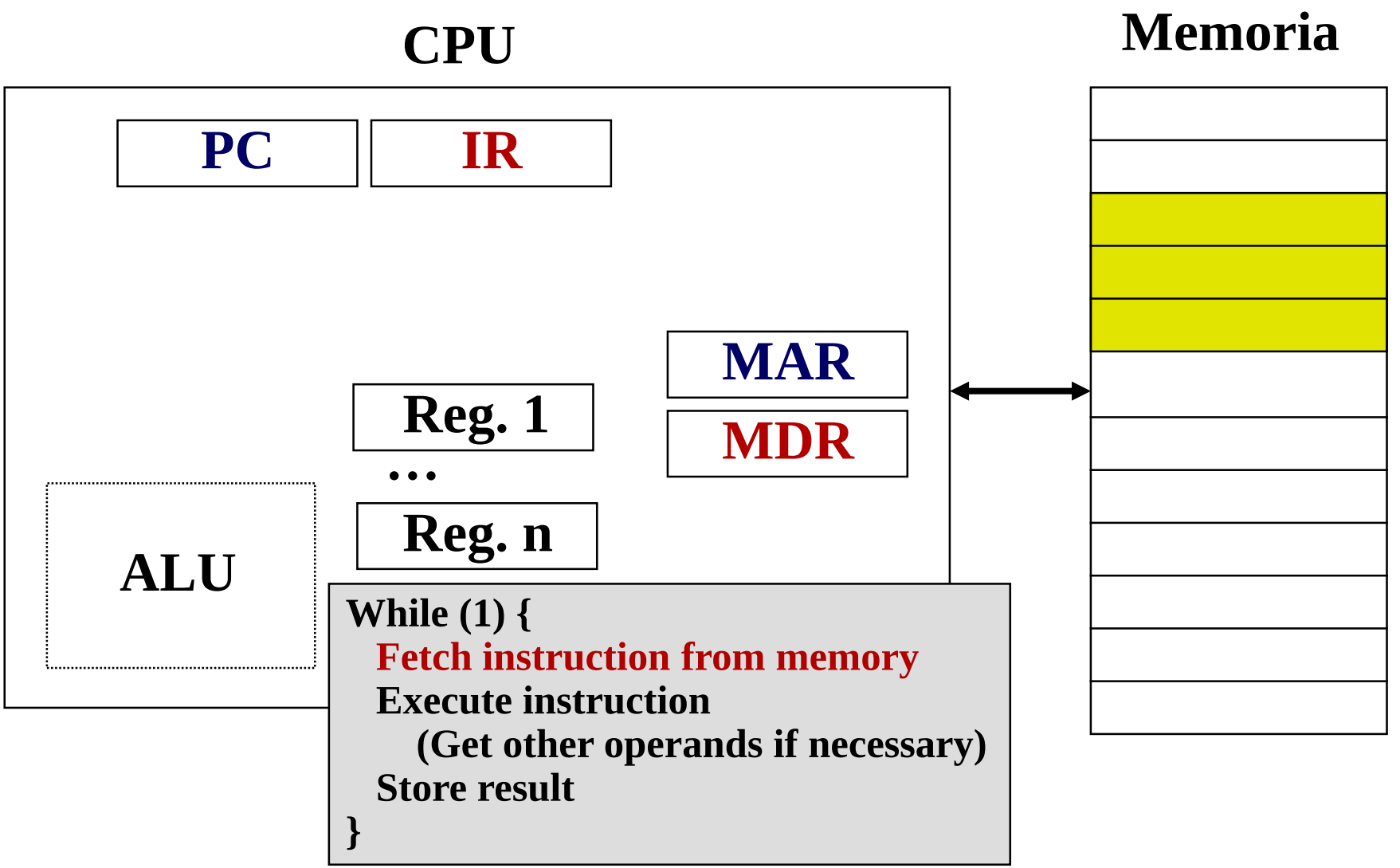
Fetch/decode/execute cycle



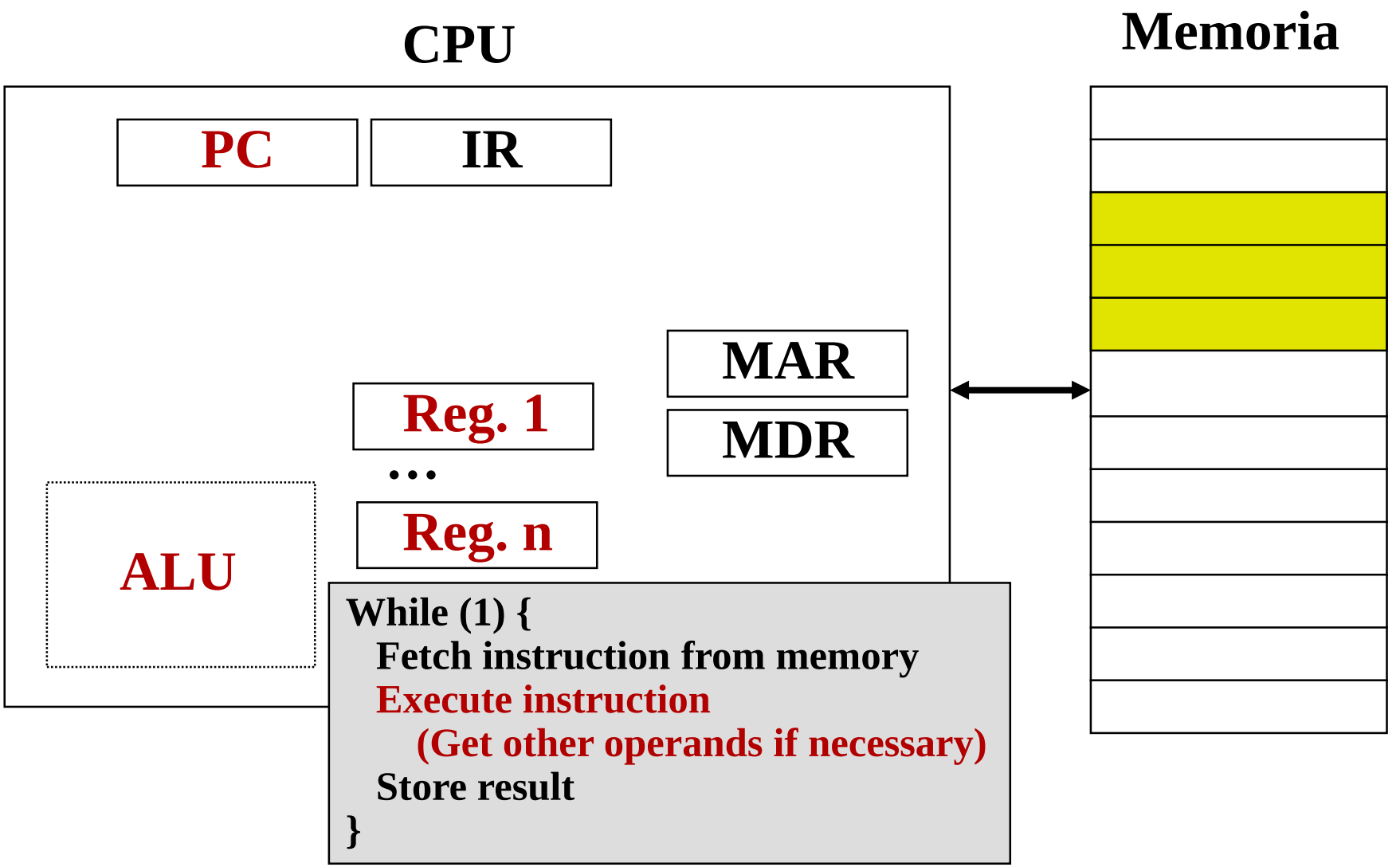
Fetch/decode/execute cycle



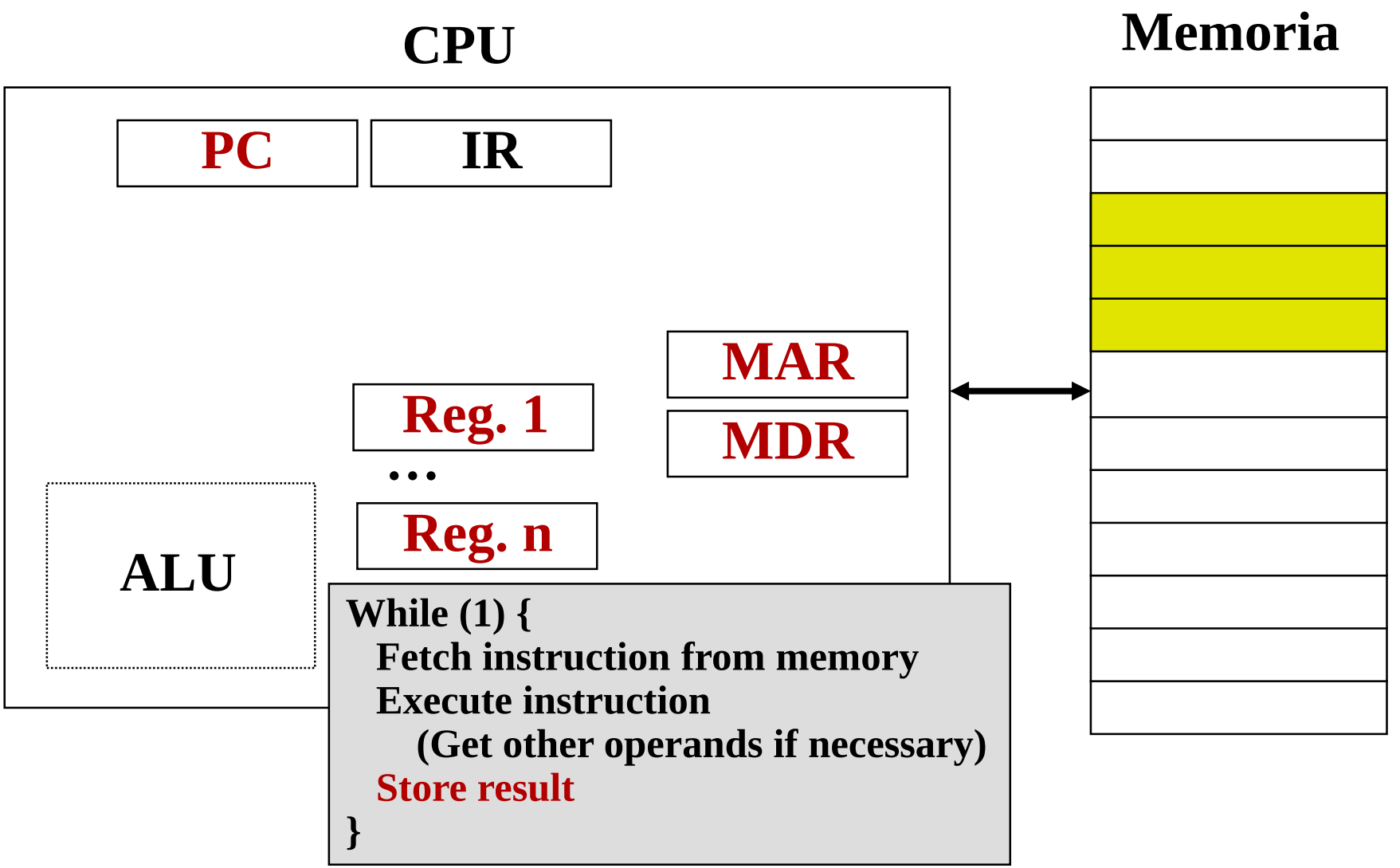
Fetch/decode/execute cycle



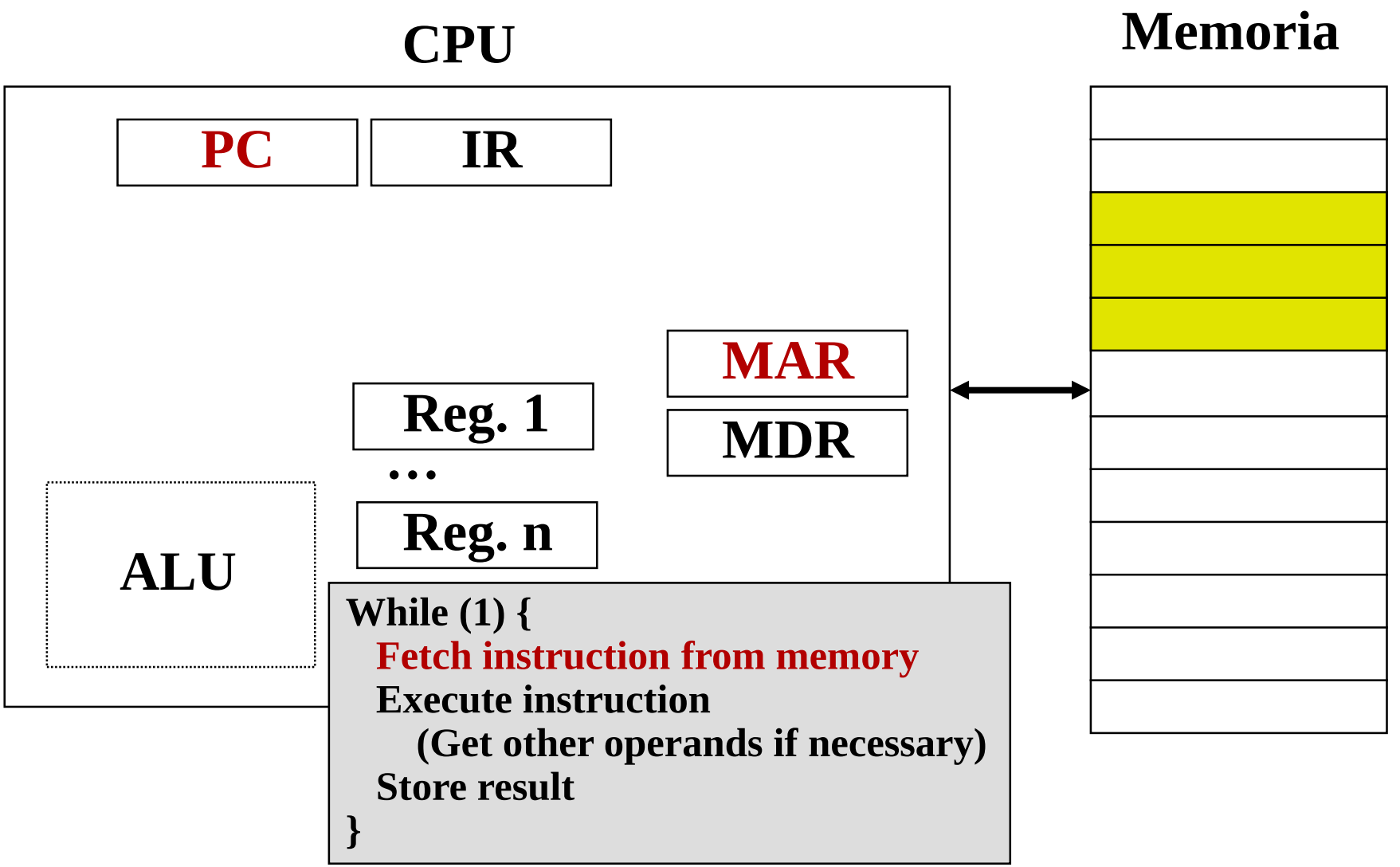
Fetch/decode/execute cycle



Fetch/decode/execute cycle



Fetch/decode/execute cycle

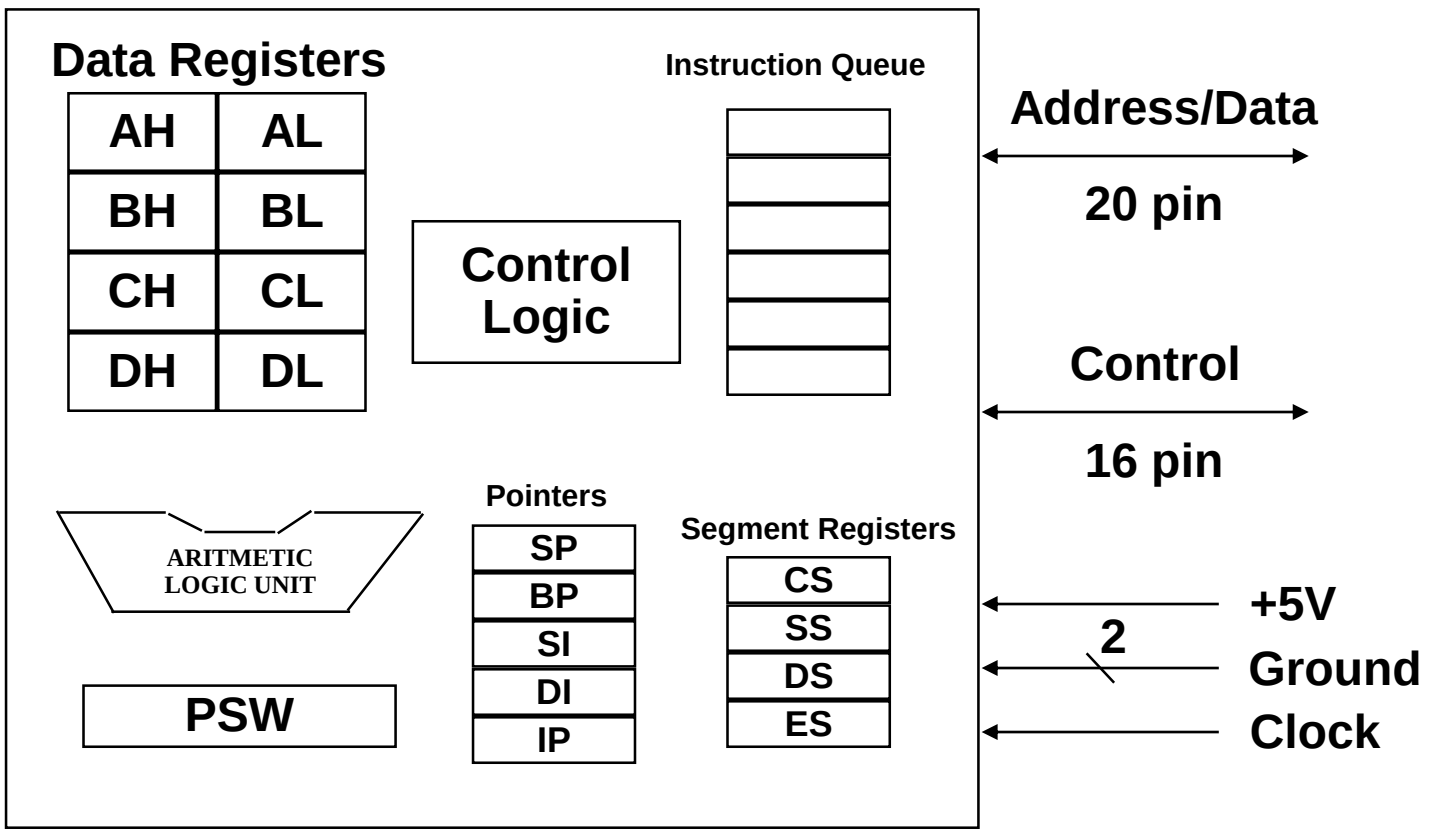


Esempio

Tutti i processori Intel (a partire dall'8086) contengono i seguenti registri (composti ciascuno da 16 bit):

- **AX, BX, CX, DX**
- **SI, DI**
- **DS, ES, CS, SS**
- **SP, BP.**

Modello architetturale della CPU 8086



Registri

Possono essere suddivisi in 3 gruppi:

- **registri di dato**
- **registri puntatore**
- **registri di segmento.**

Registri di dato

Sono AX (*Accumulator Register*), BX (*Base Register*), CX (*Count Register*) e DX (*Data Register*).

Sono utilizzati per memorizzare operandi e risultato delle operazioni.

Possono essere utilizzati come registri da 16 bit oppure come coppie di registri da 8 bit.

BX può anche essere utilizzato nel calcolo di indirizzi.

CX viene anche utilizzato come contatore da talune istruzioni.

DX contiene l'indirizzo di I/O in alcune istruzioni di I/O.

Registri puntatore

Sono IP, SP, BP, SI e DI:

- **IP (*Instruction Pointer*) contiene il puntatore alla prima istruzione da eseguire. IP non può comparire esplicitamente come operando di una istruzione.**
- **SP (*Stack Pointer*) contiene il puntatore alla testa dello stack.**
- **BP (*Base Pointer*) viene utilizzato come base per fare accesso all'interno dello stack.**
- **SI (*Source Index*) e DI (*Destination Index*) vengono utilizzati come registri indice.**

Registri di segmento

Sono CS, DS, ES e SS.

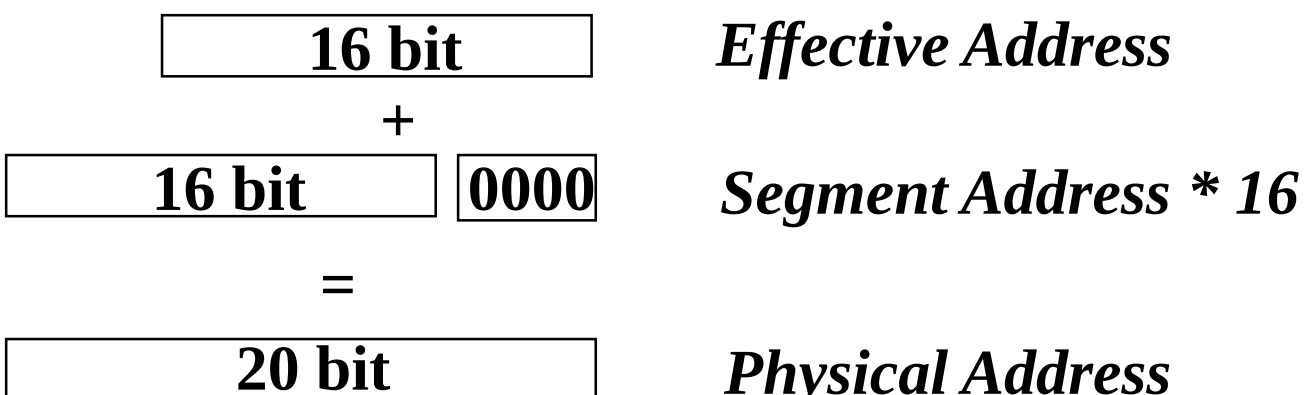
Vengono utilizzati per costruire gli indirizzi fisici con i quali fare accesso in memoria.

Contengono i puntatori all'inizio dei segmenti di codice, di dato, di dato supplementare e di stack, rispettivamente.

Calcolo degli indirizzi

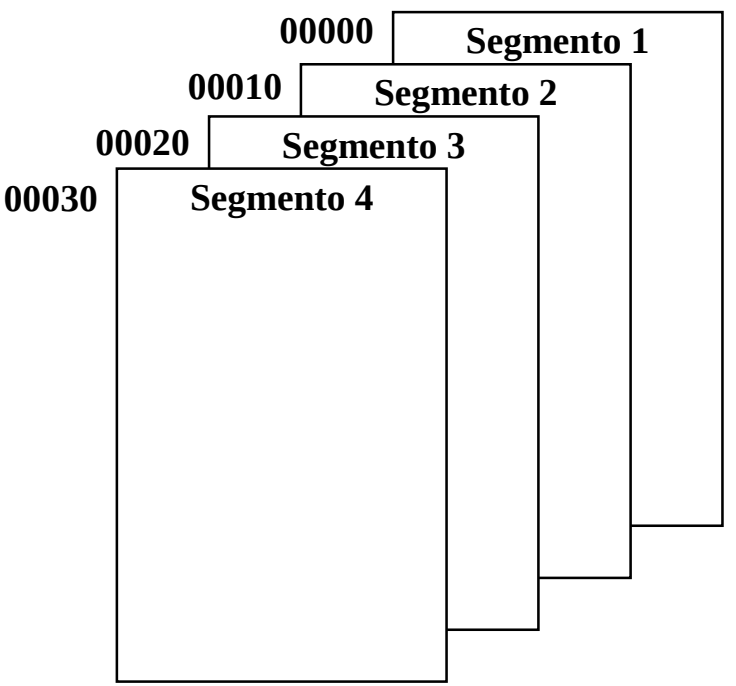
Ogni volta che l'8086 deve generare un indirizzo da mettere sull'A-bus (*physical address*), esso esegue un'operazione di somma tra il contenuto di un registro puntatore oppure di BX (*effective address* o *offset*) ed il contenuto di un registro di segmento (*segment address*).

La somma avviene dopo aver moltiplicato per 16 (shift di 4 posizioni) il contenuto del registro di segmento:



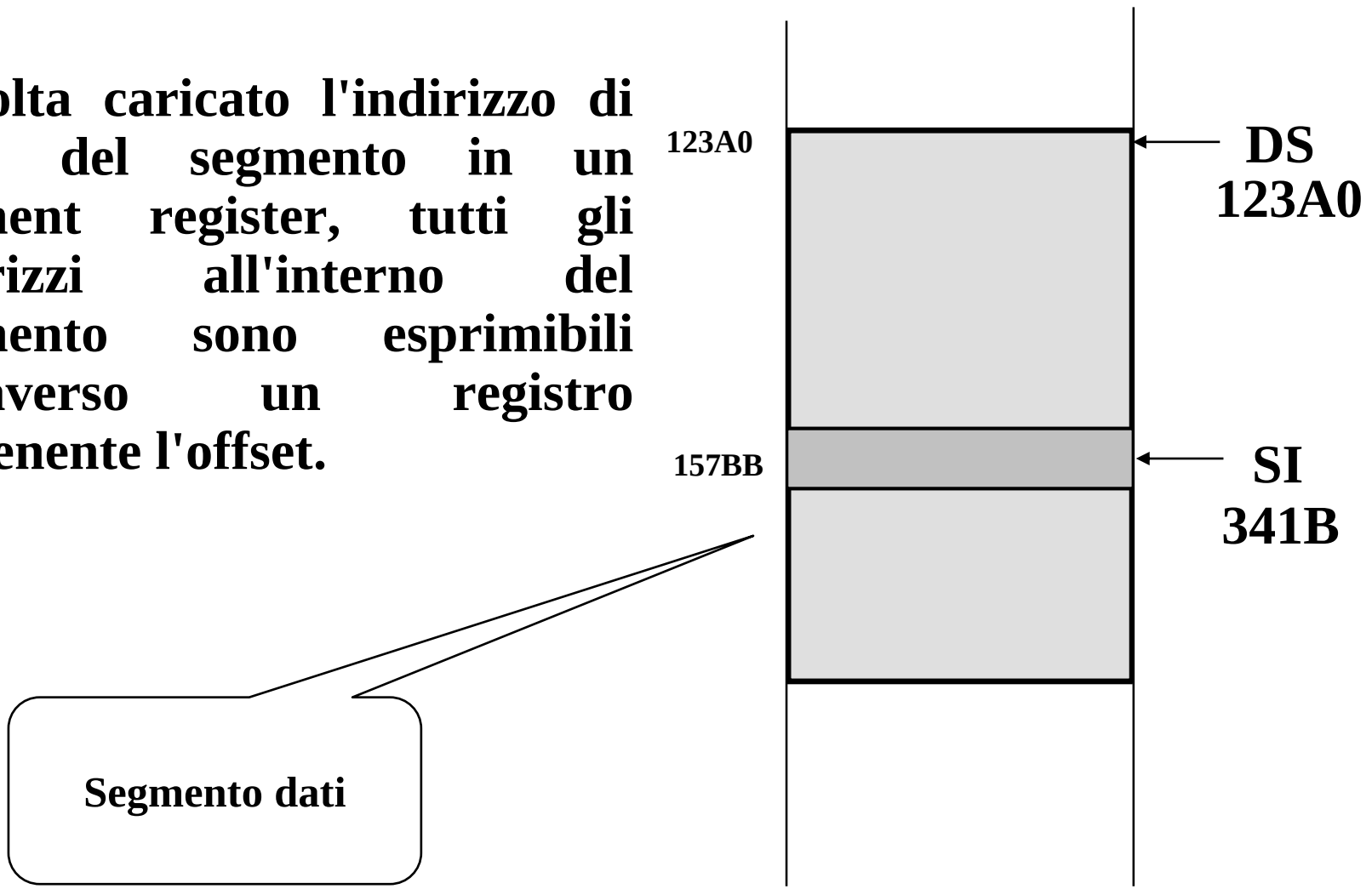
Segmenti

La memoria può essere considerata come organizzata in segmenti, ognuno di dimensione pari a 64 Kbyte. Tutti i segmenti cominciano ad indirizzi multipli di 16.

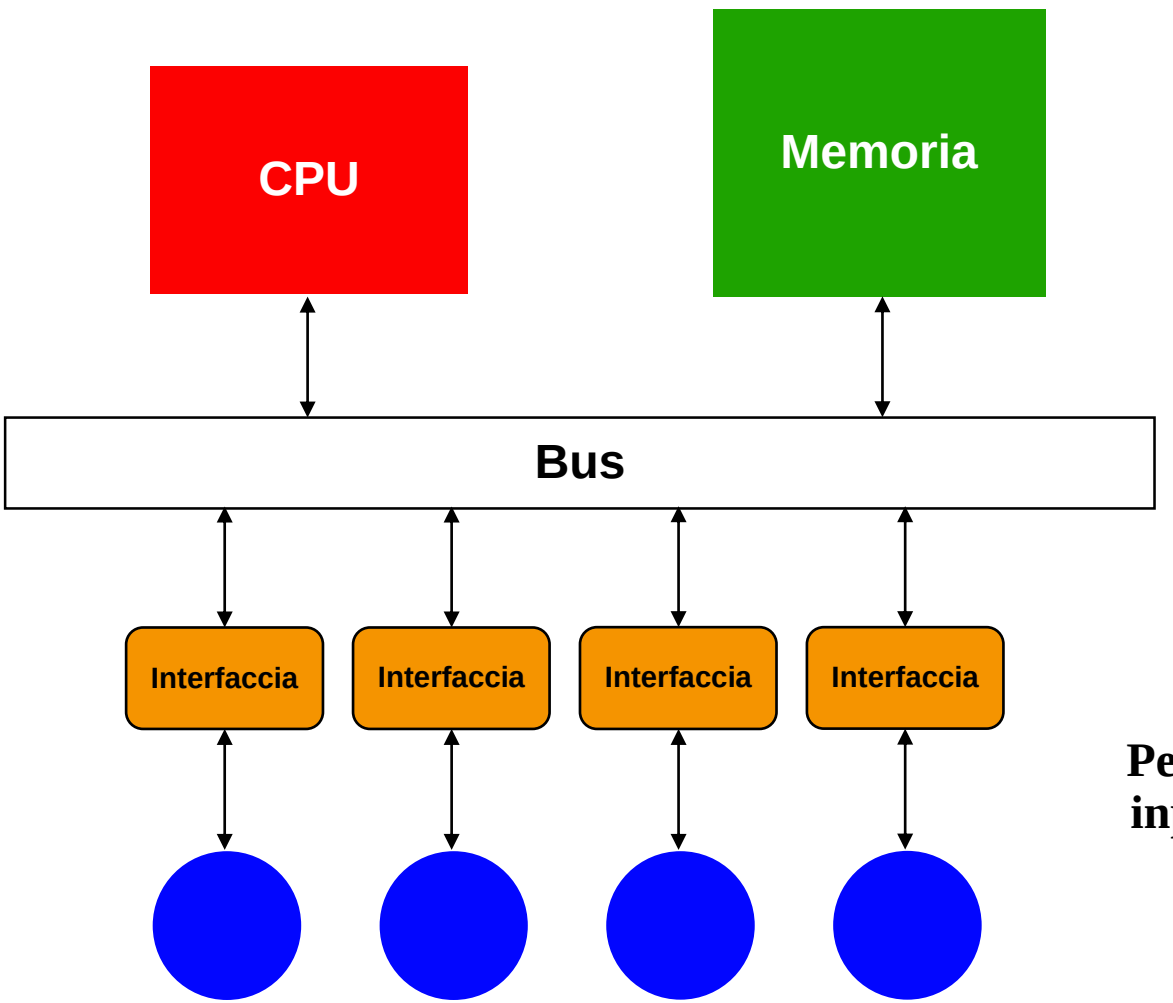


Uso dei Segment Register

Una volta caricato l'indirizzo di testa del segmento in un segment register, tutti gli indirizzi all'interno del segmento sono esprimibili attraverso un registro contenente l'offset.

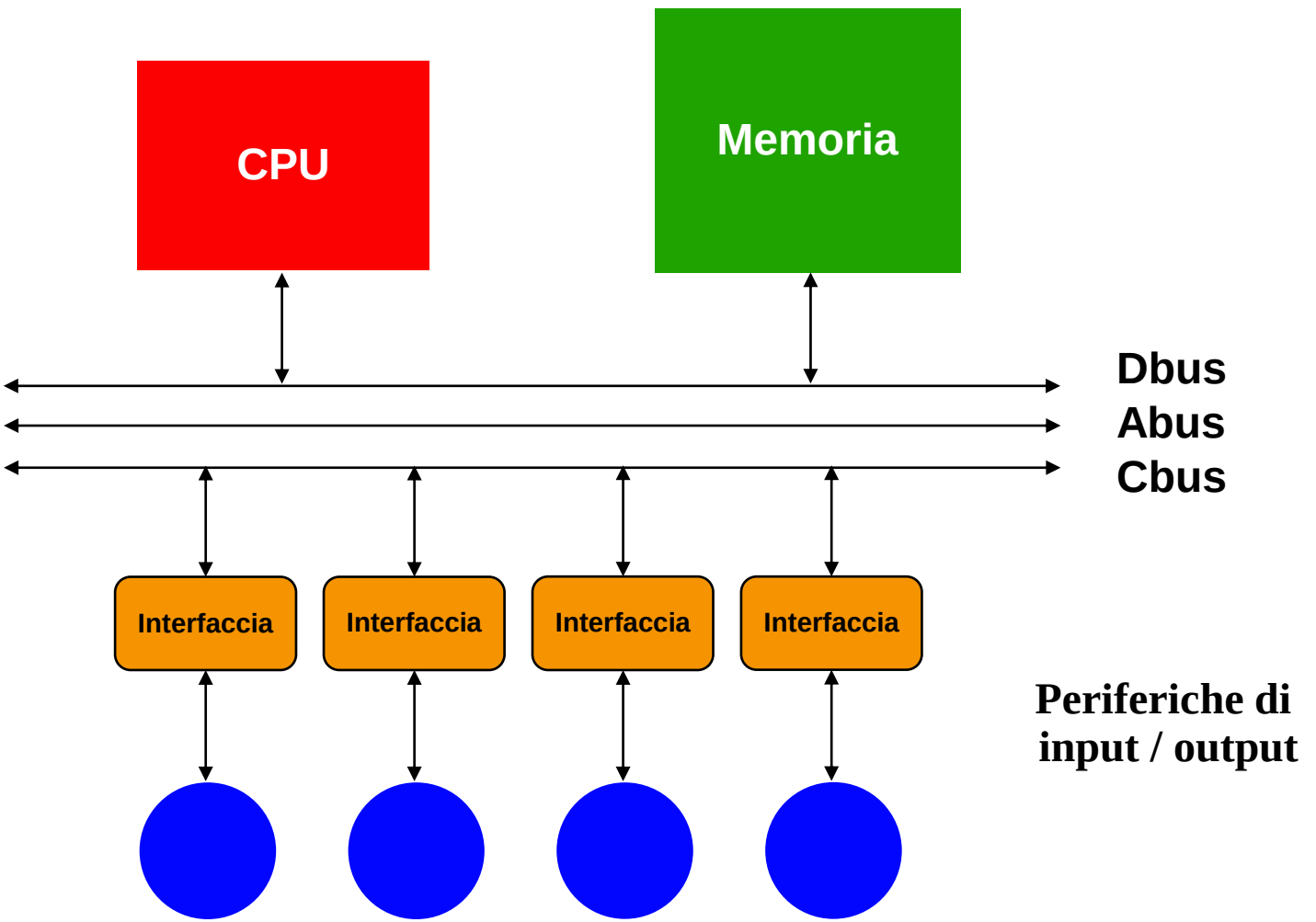


Sistema a processore

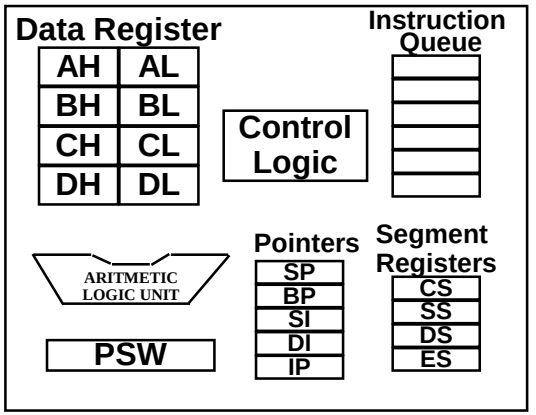


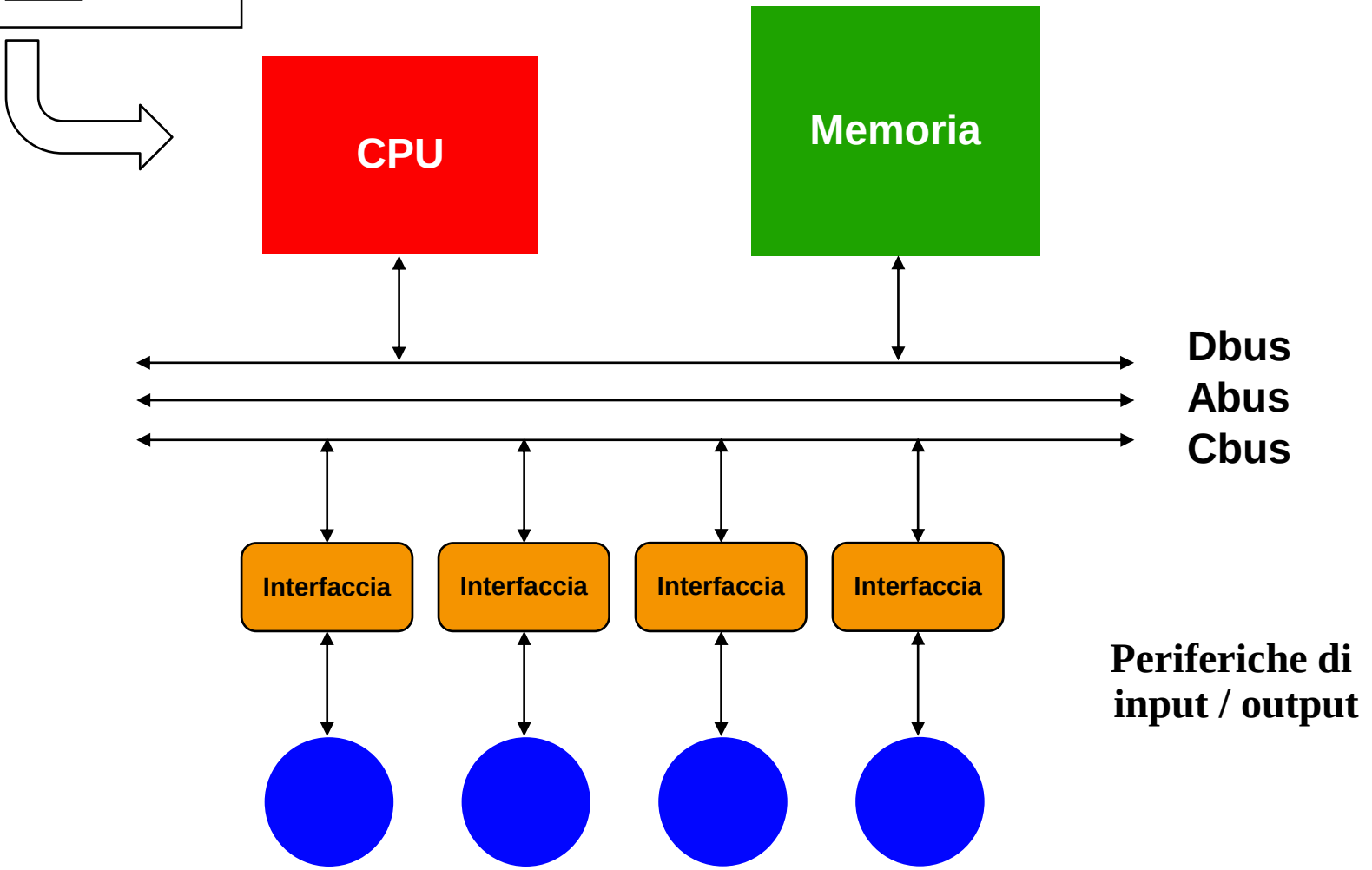
Periferiche di input / output

Sistema 8086

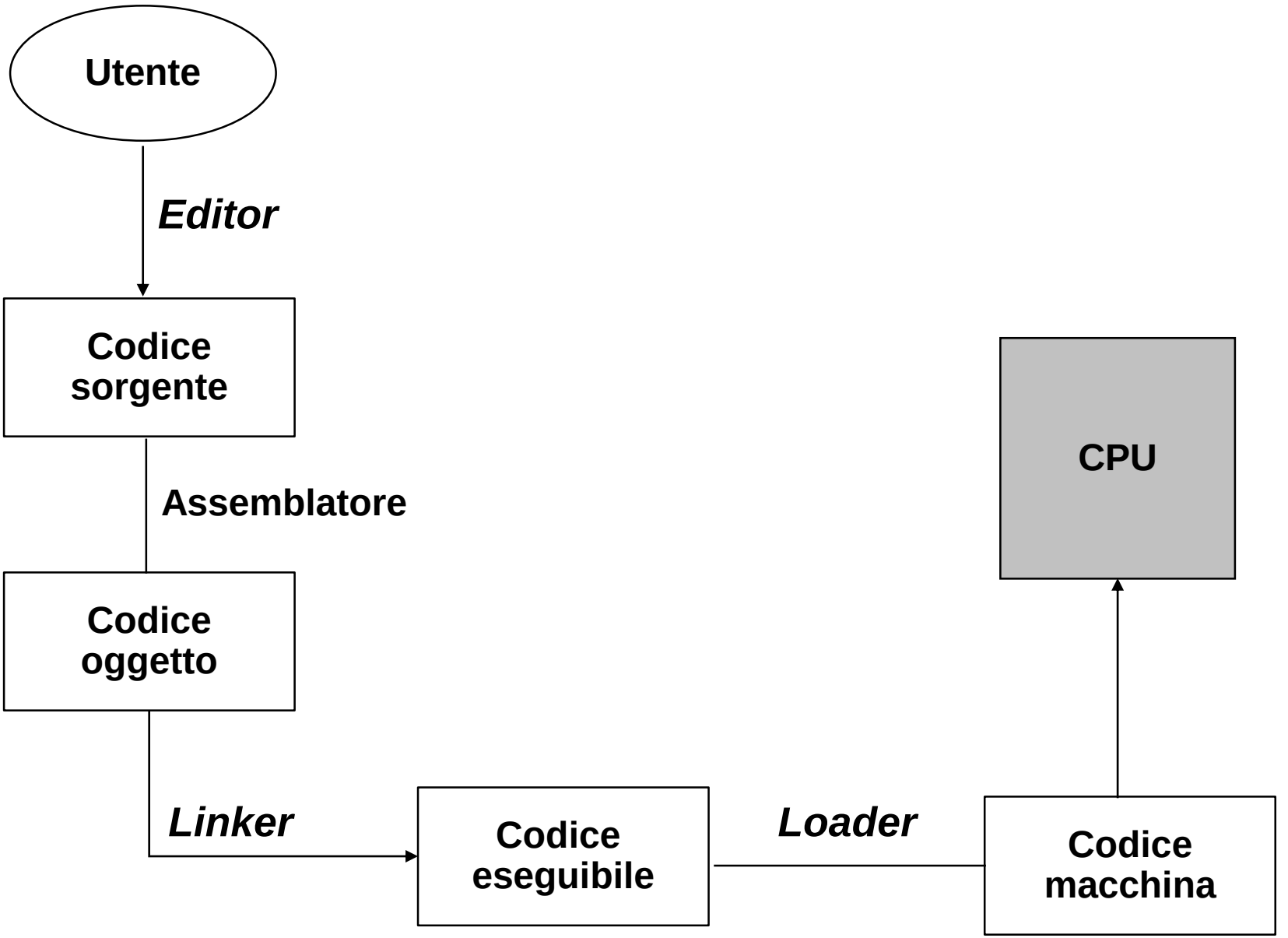


Sistema 8086





Ciclo di vita di un programma



Scrittura di un valore in un registro

```
.MODEL small
.STACK
.CODE
.STARTUP
MOV   AX, 0
.EXIT
END
```

Scrittura di un valore in una cella di memoria

```
        .MODEL small
        .STACK
        .DATA
VAR     DW    ?
        .CODE
        .STARTUP
        MOV   VAR, 0
        .EXIT
        END
```

Somma di due valori

```
        .MODEL small
        .STACK
        .DATA
OPD1    DW   10
OPD2    DW   24
RESULT  DW   ?
        .CODE
        .STARTUP
        MOV  AX, OPD1
        ADD  AX, OPD2
        MOV  RESULT, AX
        .EXIT
        END
```

Somma degli elementi di un vettore (I)

```
        .MODEL SMALL
        .STACK
        .DATA
VETT    DW   5, 7, 3, 4, 3
RESULT  DW   ?
        .CODE
        .STARTUP
        MOV  AX, 0
        ADD  AX, VETT
        ADD  AX, VETT+2
        ADD  AX, VETT+4
        ADD  AX, VETT+6
        ADD  AX, VETT+8
        MOV  RESULT, AX
        .EXIT
        END
```

Somma degli elementi di un vettore (II)

```
DIM     EQU  15
        .MODEL     small
        .STACK
        .DATA
VETT    DW 2, 5, 16, 12, 34, 7, 20, 11, 31, 44, 70, 69, 2, 4, 23
RESULT  DW   ?
        .CODE
        .STARTUP
        MOV  AX, 0              ; azzera il registro AX
        MOV  CX, DIM            ; carica in CX la dimensione
                                ; del vettore
        MOV  DI, 0              ; azzera il registro DI
```

```
lab:    ADD  AX, VETT[DI]       ; somma ad AX l'i-esimo elemento
                                ; di VETT
        ADD  DI, 2              ; passa all'elemento successivo
        DEC  CX                 ; decrementa il contatore
        CMP  CX, 0              ; confronta il contatore con 0
        JNZ  lab                ; se diverso da 0 salta
        MOV  RESULT, AX         ; altrimenti scrivi il risultato
        .EXIT
        END
```

Lettura e visualizzazione di un vettore di caratteri

```
DIM     EQU  20
        .MODEL     small
        .STACK
        .DATA
VETT    DB   DIM DUP(?)
        .CODE
        .STARTUP
        MOV  CX, DIM            ; carica in CX la dimensione
                                ; del vettore
        MOV  DI, 0              ; azzera il registro DI
        MOV  AH, 1              ; predisposizione del registro AH
```

```
lab1:   INT  21H                ; lettura di un carattere
        MOV  VETT[DI], AL       ; memorizzaz. del carattere letto
        INC  DI                 ; passa all'elemento successivo
        DEC  CX                 ; decrementa il contatore
        CMP  CX, 0              ; confronta il contatore con 0
        JNZ  lab1               ; se diverso da 0 salta
        MOV  CX, DIM
        MOV  AH, 2              ; predisposizione del registro AH
lab2:   DEC  DI                 ; passa all'elemento precedente
        MOV  DL, VETT[DI]
        INT  21H                ; visualizzazione di un carattere
        DEC  CX                 ; decrementa il contatore
        CMP  CX, 0              ; confronta il contatore con 0
        JNZ  lab2               ; se diverso da 0 salta
        .EXIT
        END
```

Ricerca del carattere minimo

```
        .MODEL      small
        .STACK
DIM     EQU  20
        .DATA
TABLE   DB   DIM DUP(?)
        .CODE
        .STARTUP
        MOV  CX, DIM
        LEA  DI, TABLE
        MOV  AH, 1                  ; lettura
lab1:   INT  21H
        MOV  [DI], AL
        INC  DI
        DEC  CX
        CMP  CX, 0
        JNE  lab1                   ; ripeti per 20 volte
        MOV  CL, 0FFH               ; inizializzazione di CL
        MOV  DI, 0
```

```
ciclo:  CMP  CL, TABLE[DI]      ; confronta con il minimo attuale
        JB   dopo
        MOV  CL, TABLE[DI]      ; memorizza il nuovo minimo
dopo:   INC  DI
        CMP  DI, DIM
        JB   ciclo
output: MOV  DL, CL
        MOV  AH, 2
        INT  21H                ; visualizzazione
        .EXIT
        END
```