L'ASSEMBLY X86

IL PROCESSORE 8086

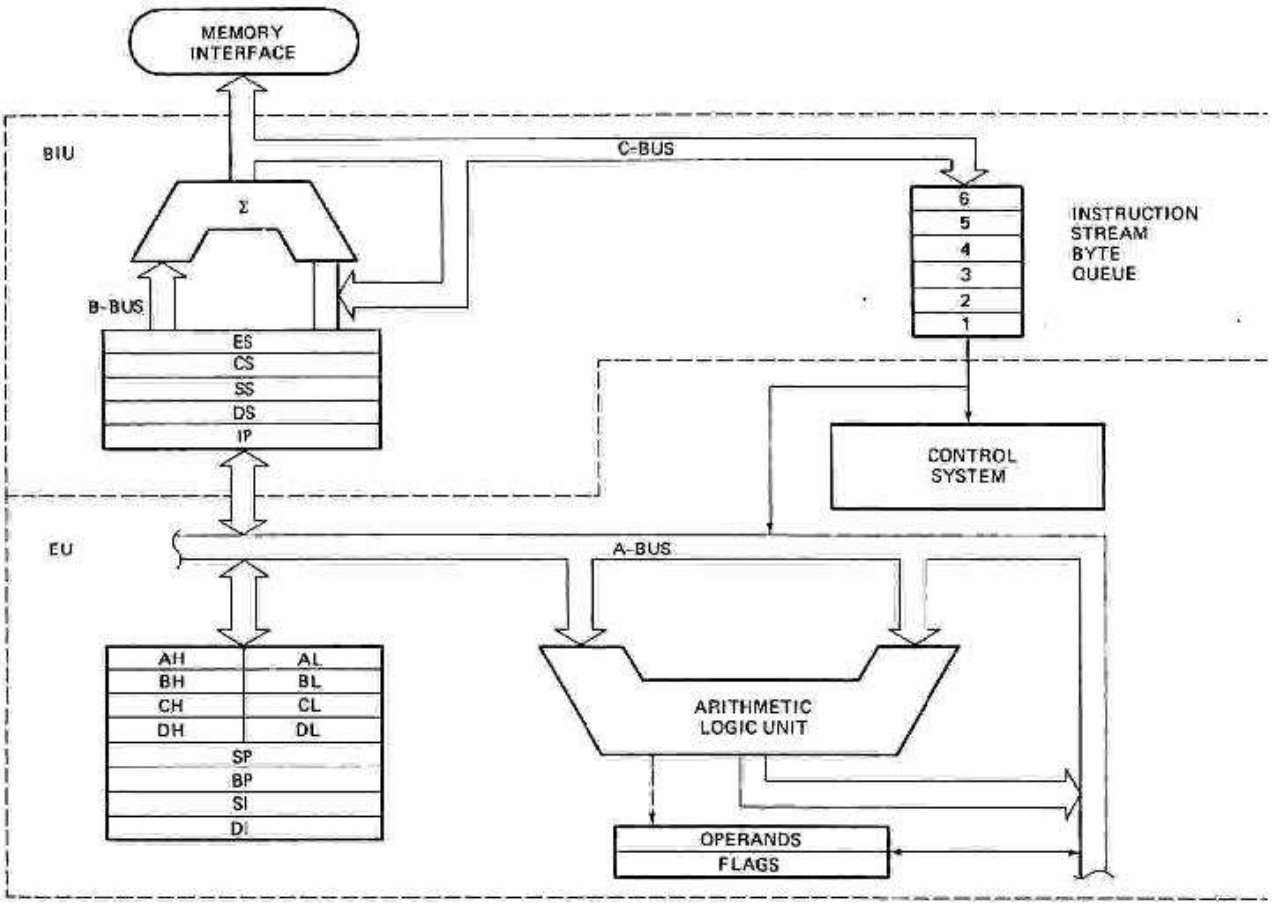


FIGURE 2-7 8086 internal block diagram. (*Intel Corp.*)

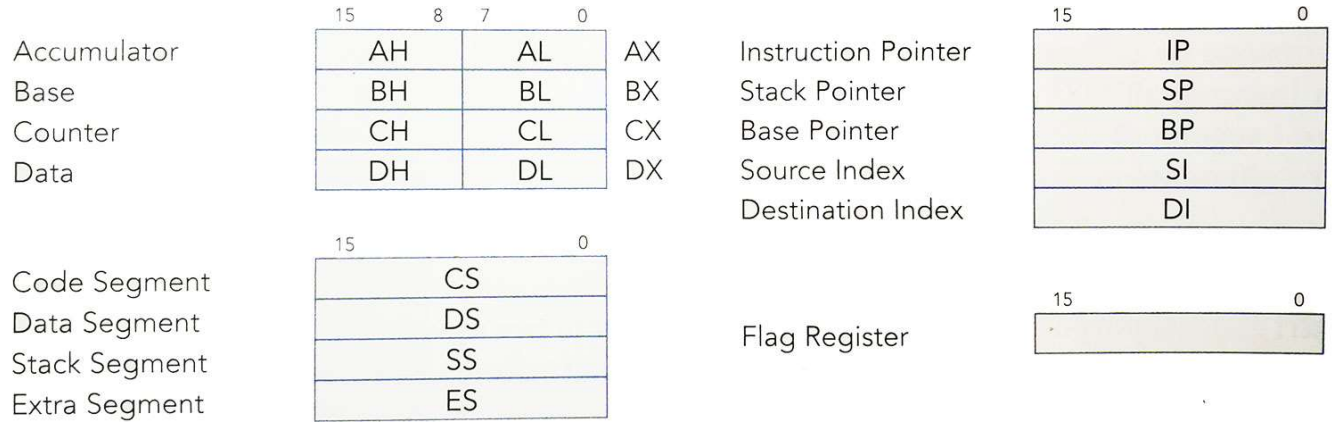
Caratteristiche del Processore 8086

- Frequenza **8 MHz**
- **BUS DATI** a 16 bit
- **BUS INDIRIZZI** a 20 bit (20 bit=>1 Mb di memoria, da 00000h a FFFFFh)
- **MULTIPLEXAGGIO** bus dati e indirizzi : dati ed indirizzi viaggiano sulle stesse linee fisiche,ma in tempi diversi
- **14 Registri interni** a 16 bit (4 utilizzabili come coppia di registri a 8 bit)
- Struttura a **Pipeline** (EU e BIU) : mentre la EU sta eseguendo un'istruzione la BIU ne precarica altre nella coda di *prefetch*
- **Set di 70 istruzioni** di base: operazioni aritmetiche (binarie e BCD) su numeri con e senza segno a 8 e 16 Bit, manipolazione di stringhe, operazioni logiche, istruzioni di salto

X86 O PC COMPATIBILE

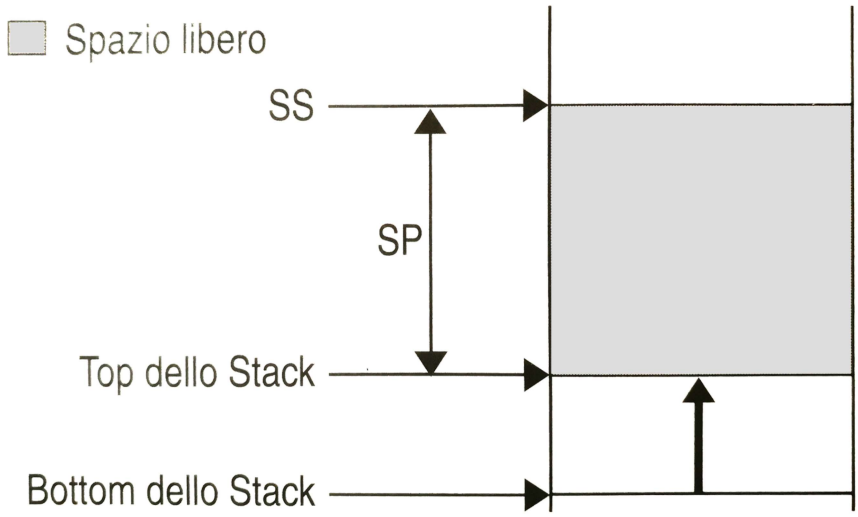
Il **Modello di Programmazione**, cioè quella parte dell'architettura del microprocessore accessibile al programmatore composta da ***Registri*** e ***Set di Istruzioni***, del ***processore 8086*** è chiamato **x86** o **PC compatibile**.

I principali **Registri**, tutti a 16 bits, del *processore 8086* sono:



Il **Registro Accumulatore** (**AX**) è quello che viene generalmente usato per le operazioni *logiche* ed *aritmentiche* sui dati.
Il **Registro Base** (**BX**) viene utilizzato principalmente per memorizzare l'indirizzo di partenza di una cella di memoria; a tale indirizzo viene sommato un OFFSET in modo da ottenere un indirizzo finale per identificare una cella di memoria sulla quale leggere o scrivere.
Il **Registro Contatore** (**CX**) viene usato per effettuare il conteggio all'interni dei cicli, per operazioni di rotazione e scorrimento o per le istruzioni di gestione dell'I/O.
Il **Registro Dati** (**DX**) è utilizzato in modo analogo all'*Accumulatore* o per effettuate la lettura e la scrittura sui dispositivi I/O.

La **Stack** è un'area di memoria della memoria centrale dove i dati vengono accatastati.
I Registri legati alla **Stack** sono lo **Stack Pointer** (**SP**) e lo **Stack Segment** (**SS**).
l'**SS** è l'*indirizzo del segmento di memoria* e lo **SP** l'*offset della cima dello Stack*:



L'8086 è costituito da due sottoinsiemi che operano in modo parzialmente indipendente a asincrono:

- **EU** (***Execution Unit***) costituisce la parte della CPU che *elabora*
  – Registri ***General Pourpose*** (**AX**, **BX**, **CX**, **DX**)
  – Registri ***Speciali*** (**PSW** e **Flag**)
  – Unità di ***controllo*** (**EU control**)
  – **ALU**
- **BIU** (**BUS Interface Unit**) gestisce l'*indirizzamento*, il prelievo dei dati e delle istruzioni dalla memoria e gestisce il colloquio con i dispositivi esterni
  – Logica di ***controllo dei BUS***
  – Registri di ***Segmento*** (**CS**, **DS**, **SS**, **ES**)
  – Registro ***Program Counter*** (**IP**)
  – Registri ***Puntatore*** (**BP**, **SP**, **DI**, **SI**)

LINGUAGGIO ASSEMBLER

Il linguaggio Assembly si distingue dagli altri linguaggi perchè possiede una corrispondenza *uno a uno* con il **linguaggio macchina**, quindi per ogni ***codice operativo*** delle istruzioni di una CPU esiste una corrispondente ***istruzione in Assembly***.

Codice mnemonico Assembly	Linguaggio macchina esadecimale	Linguaggio macchina binario
XOR AX,AX	33 C0	00110011 11000000
INC AX	40	01000000
ADD AL,01	41	01000001

Per il motivo che è strettamente legato all'architettura del processore l'**Assembly non è un linguaggio portabile**.

Il **set di istruzioni Assembler** Intel 8086 può essere suddiviso nelle seguenti classi:
– Istruzioni di *trasferimento*: **mov**, **push**, **pop**, …
– Istruzioni *aritmetiche*: **add**, **sub**, **cmp**, **mul**, **div**, …
– Istruzioni *logiche*: **and**, **or**, **xor**, **not**, …
– Istruzioni di *manipolazione di bit*: **shl**, **shr**, …
– Istruzioni di *controllo*: **jmp**, **call**, **ret**, **jg**, **jge**, **loop**, …
– Istruzioni di *input/output*: **in**, **out**, …
– Istruzioni di *manipolazione di stringhe*: **movs**, **stos**, **lods**, …
– Istruzioni di *controllo dello stato della macchina*: **hlt**, **wait**, …

Un **PROGRAMMA ASSEMBLY** deve, per prima cosa, dichiarare come viene utilizzata la memoria e come viene suddivisa nei 4 segmenti fondamentali (*Dati* **DS**, *Stack* **SS**, *Codice* **CS**, *Extra* **ES**).
I Segmenti vengono dichiarati attraversi **Direttive** (*standard* o *semplificate*).
La ***Struttura tipica di un programma*** Assembly è la seguente:

```
TITLE Titolo Programma
.MODEL small            ; identifica la dimensione dei Segmenti in uso
.STACK 100h             ; valore inizializzato puntato da SS

.DATA                   ; Data Segment: dati puntati da DS
Dichiarazione variabili

.CODE                   ; Code Segment: istruzioni codice puntato da CS
.STARTUP                ; direttiva che determina i valori per DS e ES

    ; servizio DOS di chiusura del programma
    mov ah,4ch
    int 21h
    END
```

```
title Stampa stringa

data segment
; creo le variabili
stringa db "Premi un tasto per uscire…$"
hello db "Hello World!",0Ah,0Dh,"$" ; 0Ah => LF (Line Feed) avanza di una riga
; 0Dh => CR (Carriage Return) riporta ad inizio riga
; in sequenza 0Ah e 0Dh significano 'a capo'
ends
```

```
stack segment
ends

code segment
start:
; setto regitri di segmento:
mov ax, data
mov ds, ax

; uso INT per stampare la stringa "Hello World!"
lea dx, hello
mov ah, 9
int 21h ; output stringa all'indirizzo ds:dx (dx dove ho meso la stringa)

; vado a capo

; uso INT per stampare la stringa "Premi un tasto per uscire…"
lea dx, stringa
mov ah, 9
int 21h ; output stringa all'indirizzo ds:dx (dx dove ho meso la stringa)

; attendo la pressione di un tasto….
mov ah, 1
int 21h

mov ax, 4ch ; esco dal programma.
int 21h
ends

end start
```

Nella **Dichiarazione delle variabili** oltre al *nome* occorre definire il *tipo di memoria* ed il *contenuto*, es:

[nome] [tipo] [contenuto]

Messaggio **DB** **"Salve Mondo",13,10,'$'**

dove ***Messaggio*** è il *nome della variabile*, **DB** il *tipo di memoria* allocata (Byte) e **"Salve Mondo",13,10,'$'** il valore della stringa memorizzata (è il concatenamento di "Salve Mondo" + \r + \n + terminatore stringa '$', obbligatorio in caso di stringhe).

Il **FORMATO DELLE ISTRUZIONI** di un programma Assembly si sviluppa su 4 colonne:

[Etichetta:] Istruzione/Direttiva [Operando, Operando] [;Commento]

L'**Etichetta** (**Label**) consente di dare un *nome simbolico* alle istruzioni da utilizzare come operando nei salti o nelle procedure

Istruzioni e **Direttive** possono essere scritte in maiuscolo o in minuscolo. Le ***Istruzioni*** sono parole chiave che identificano le oprazioni da eseguire (esempio MOV, ADD). Le ***Direttive***sono istruzioni particolari che dialogano con l'assemblatore e iniziano con il carattere punto (esempio .CODE)

Gli **Operandi** sono la combinazione di 3 tipi di dati: **registri** (esempio AX, CX), ***cella di memoria*** (esempio puntatore [SI], nome variabile), ***dato immediato*** (valore numerico).

I **Commenti** servono per rendere più leggibile il programma, soprattutto nei linguaggi poco leggibili come l'Assembly, e sono preceduti dal ***punto e virgola*** **;**

Il programma deve quindi essere salvato con estensione **.asm**, ed il suo ***nome*** deve avere una **lunghezza massima di 8 caratteri**.

***nome*.asm**

(se si utilizza Turbo Assembler deve essere posizionato all'interno della cartella **TASM/BIN**).

I **METODI DI INDIRIZZAMENTO** sono i sistemi utilizzati dalle istruzioni per accedere agli operandi.
Le operazioni svolte dalle istruzioni possono avere **1 operando** o **2 operandi**.
Ogni istruzione di Assembly deve specificare alla CPU il ***metodo di accesso*** ai dati (dipendente dal metodo di indirizzamento o del registro sul quale operano) attraverso quello che è il ***codice operativo dell'istruzione*** (codice macchina).
La CPU non è in grado di trasferire direttamente i dati da memoria a memoria; il dato deve **sempre passare per un registro**.

I vari *tipi di indirizzamento* possibile sono:
Indirizzamento Immediato, l'operando dell'istruzione è contenuto nel codice operativo dell'istruzione stessa ed è quindi una ***costante*** (esempio MOV AX, **05**)

Indirizzamento a Registro, l'operando dell'istruzione è contenuto all'interno di un ***registro*** (esempio MOV **AX**, **BX**)

Indirizzamento Diretto, in questa modalità l'operando dell'istruzione è contenuto a partire dalla cella di memoria il cui indirizzo è contenuto nel codice dell'istruzione stessa. Il dato è quindi una ***variabile di posizione costante*** (esempio NEG **variabile1**)

Indirizzamento Indiretto, l'operando dell'istruzione è contenuto a partire dalla cella di memoria il cui indirizzo è contenuto in un registro puntatore. Il dato è quindi una ***variabile di posizione variabile*** (esempio MOV AX, **[BX]**)

Indirizzamento Indicizzato, l'operando dell'istruzione è contenuto a partire dalla cella di memoria il cui indirizzo è calcolato dalla somma di 2 registri in cui il primo è considerato *puntatore alla base di un blocco di memoria* mentre il secondo è considerato *spiazzamento*, cioè spostamento all'interno del blocco di memoria. Il calcolo di questa somma tra registri è effettuato internamente, pertanto il dato è una ***variabile di posizione doppiamente variabile***perchè può variare sia il contenuto del registro puntatore sia quello dell'indice (esempio MOV **[BX+SI]**, CX o MOV **ES:[BX]**, AL)

Indirizzamento Implicito, l'operando non viene specificato in alcun modo in quanto ***implicitamente dichiarato nel codice operativo*** dell'istruzione stessa.