Corso di Architettura degli Elaboratori
Modulo di Assembly

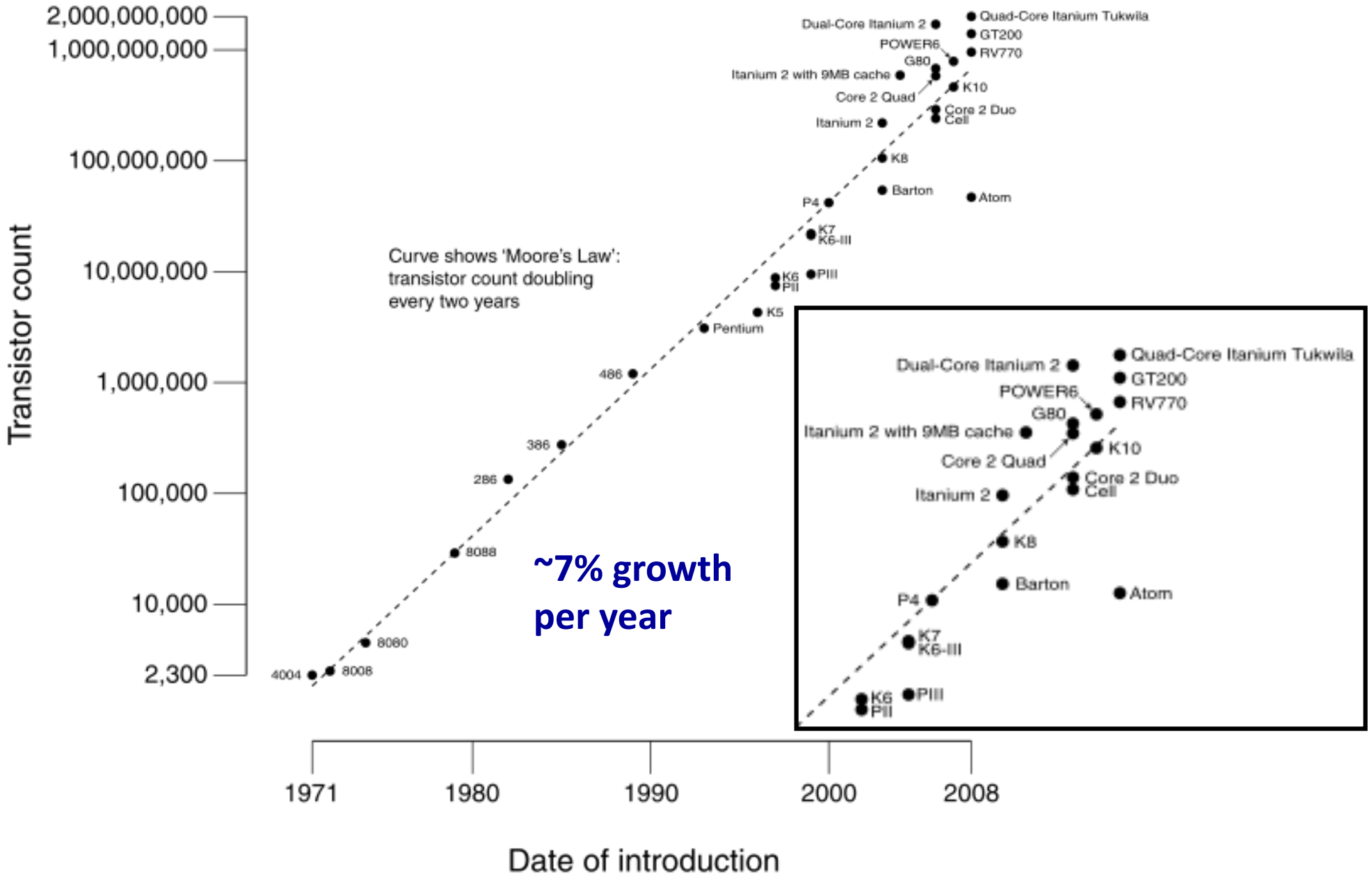
ARCHITETTURA 8088

Bruno Iafelice

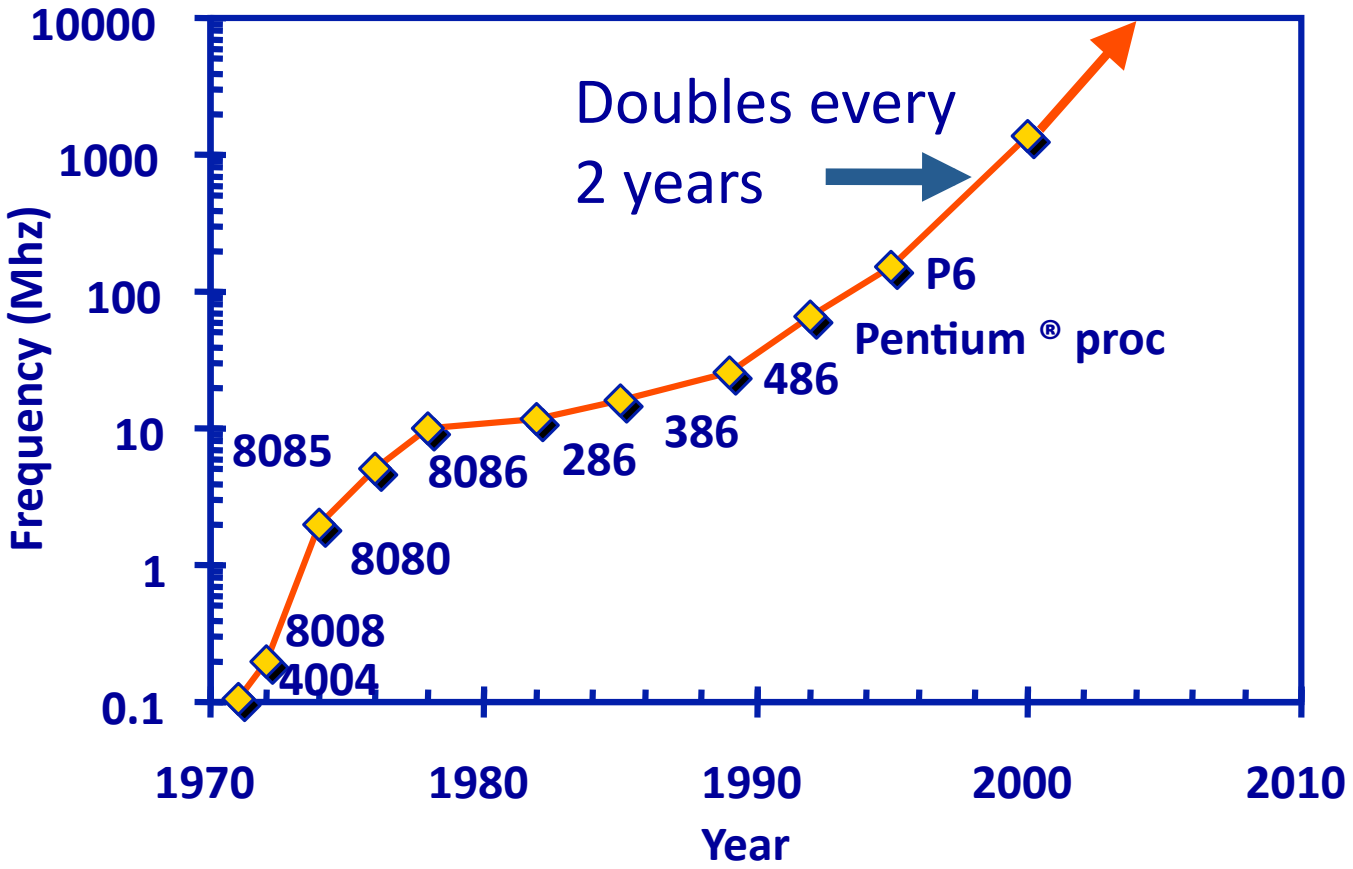
Università di Bologna

iafelice at cs(dot)unibo(dot)it

Evoluzione: crescente integrazione



Aumento della frequenza: numero di operazioni per unità di tempo



Courtesy, Intel

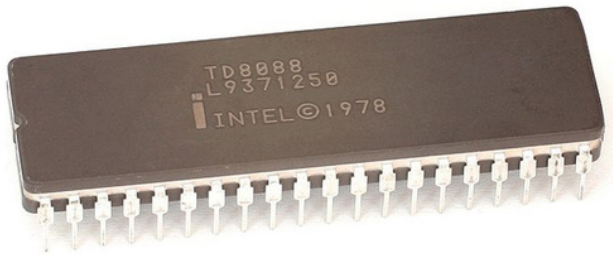
Intel processors

CPU	*Year*	*Data*	*Memory*	*MIPS*
4004	1971	4	1K	
8008	1972	8	16K	
8080	1974	8	64K	
8088	1980	8	1M	.33
80286	1982	16	1M	3
80386	1985	32	4G	11
80486	1989	32	4G	41
Pentium	1993	64	4G	111

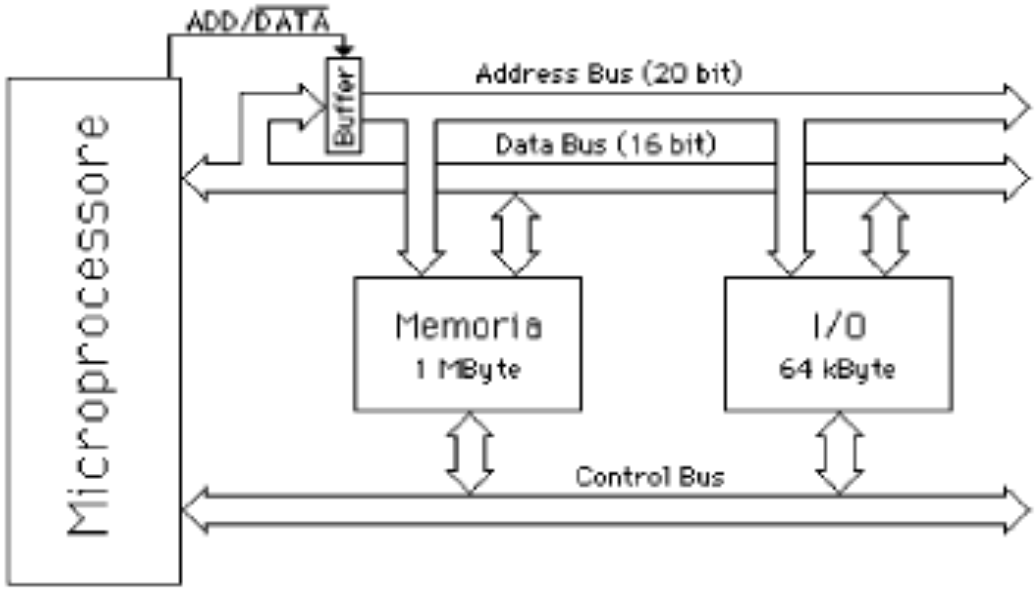
Million Instructions Per Second

Architettura 8088 (8086)

- microprocessori Intel della terza generazione
- progetto del 1978/79
- address bus: 20 bit
- memoria 1M byte
- data bus: 8 bit per l'8088, 16 bit per l'8086
- identico formato delle istruzioni



Caratteristiche	8086/8088	80286	80386
address bus	20 bit	24 bit	32 bit
data bus	16 / 8 bit	16 bit	32 bit
indirizzamento	1M byte	16M byte	4G byte
		1G byte virtual	64T byte virtual
registri	16 bit	16 bit	32 bit
piedini	40	68	132
computer IBM	25,30 / PC,XT	AT,50,60	80



- Multiplex per contenere il numero di piedini
- Ogni trasferimento: 2 cicli di bus (indirizzo - dato)

Memoria principale

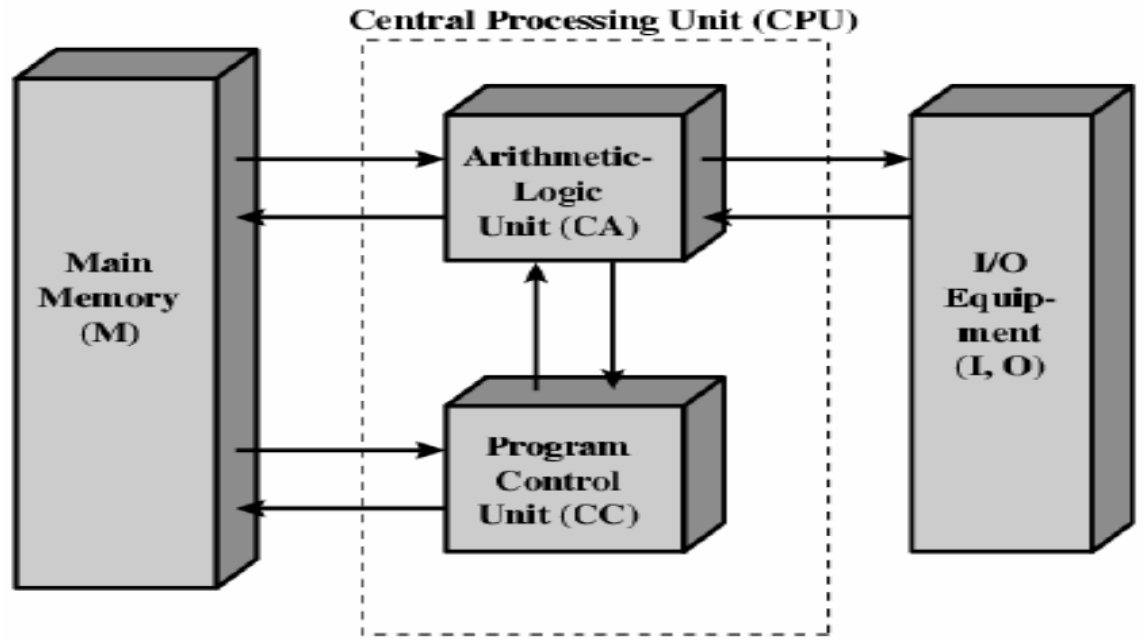
- 1M byte di memoria
- 2^{20} = 1.048.576 locazioni di memoria di 8 bit
- il primo byte ha indirizzo 0
- l'ultimo byte ha indirizzo 0FFFFFh
- Accesso contemporaneo al massimo a 4 segmenti di memoria di 64k byte ciascuno (max 256k byte)

I/O

L'8088 gestisce l'I/O tramite:

- indirizzi di memoria
- indirizzi di I/O, distinti dagli indirizzi di memoria, in uno spazio di indirizzamento di 64k byte (da 0 a 0FFFFh)
- Gli indirizzi di I/O possono essere utilizzati esclusivamente nelle istruzioni di I/O: `IN` e `OUT`

von Neumann architecture

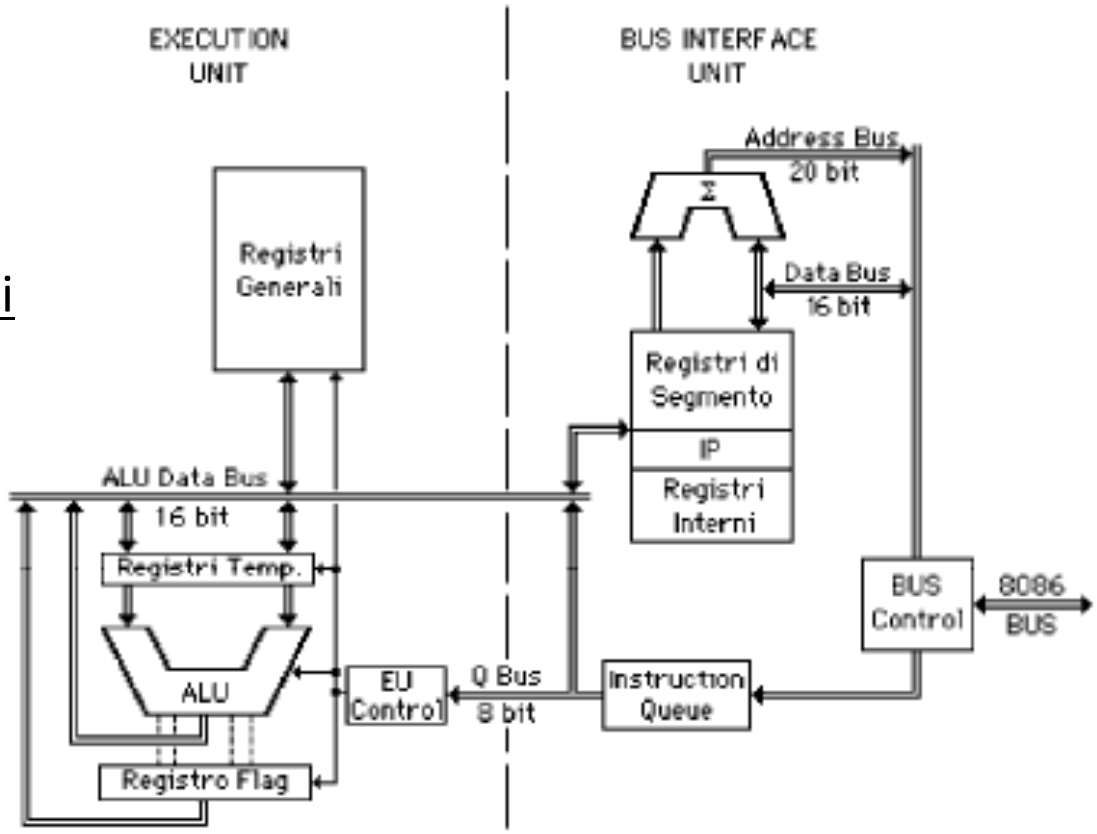


- Code and data stored together
- Arithmetic unit
- Load and control unit
- I/O

CPU

- Execution Unit (EU):
  - esegue le istruzioni (fase di execute)
- Bus Interface Unit (BIU):
  - rintraccia le istruzioni (fase di fetch)
  - legge gli operandi
  - scrive i risultati

- Le due unità operano in maniera indipendente e concorrente.



CPU - BIU

- **calcola gli indirizzi reali a 20 bit sommando, in un sommatore dedicato, l'indirizzo del segmento e l'offset (entrambi a 16 bit)**
- **esegue il trasferimento dei dati da e verso l'EU**
- **carica le istruzioni nella coda delle istruzioni (prefetch)**

Registri

- l’8088 ha 14 registri
- sono lo spazio di lavoro delle istruzioni
- e’ uno spazio di lavoro TEMPORANEO
- non adatto a contenere dati in forma definitiva
- e’ uno spazio di lavoro interno alla CPU (gerarchia di memorie.....)

Registri GENERALI

PUNTATORI e INDICI

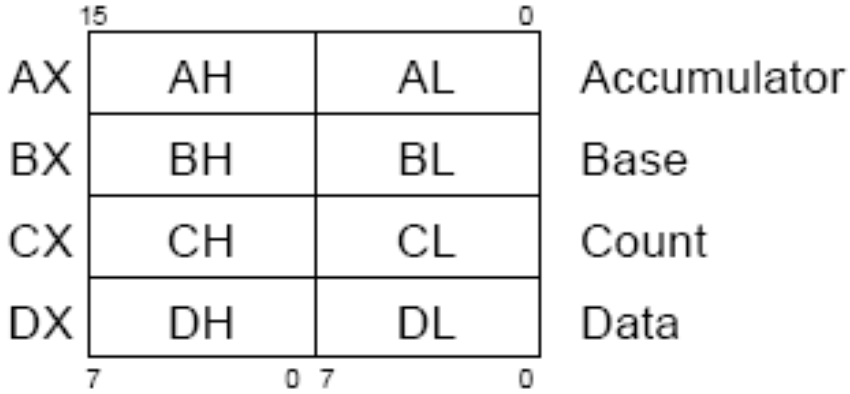
Registri di SEGMENTO

Program counter (PC)

Registro dei Flag

Registri GENERALI
(Registri di Dati)

AX, BX, CX, DX



Utilizzabili come:

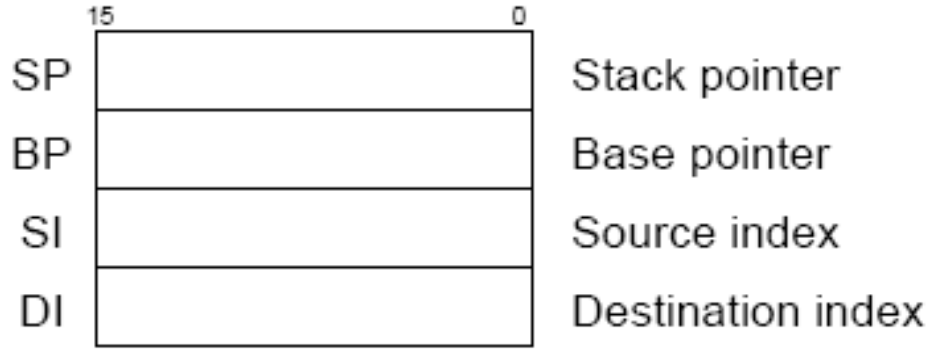
- registri a 16 bit (AX)
- registri a 8 bit (AH e AL - AHigh e ALow)

- AX registro Accumulatore, è usato per raccogliere il risultato di molte operazioni
- BX registro di Base, è usato per memorizzare indirizzi di memoria
- CX registro Contatore, usato ad esempio come contatore di cicli
- DX registro Dati, usato con AX per contenere i dati

Tutti i registri possono essere usati come operando o come registri temporanei.

PUNTATORI e INDICI

DI, SI



Utilizzi:

- puntatori
- indici nei cicli (es. ciclo FOR) o negli IF
- come variabili temporanee

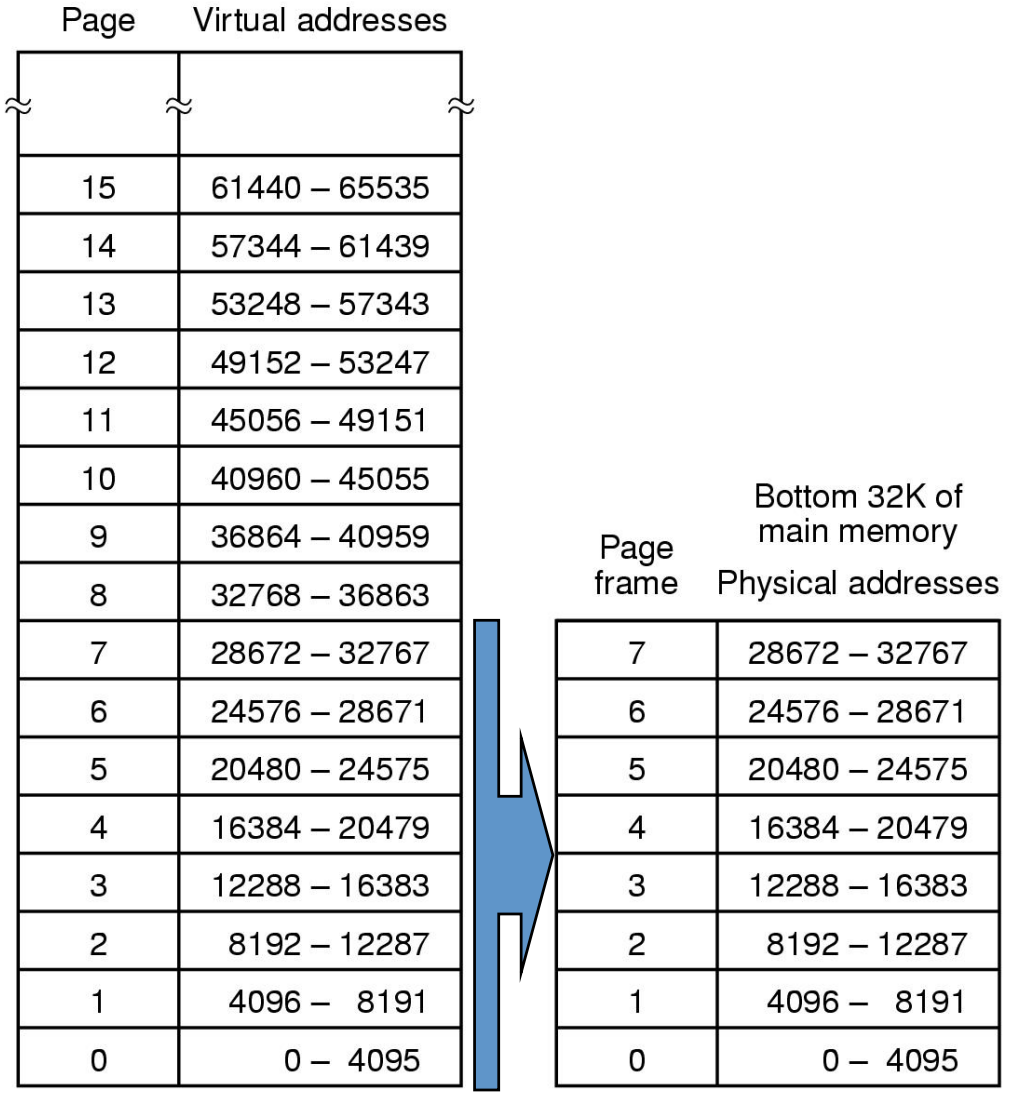
Sono a 16bit

ORGANIZZAZIONE DELLA MEMORIA E SEGMENTI

Memoria – Memoria virtuale

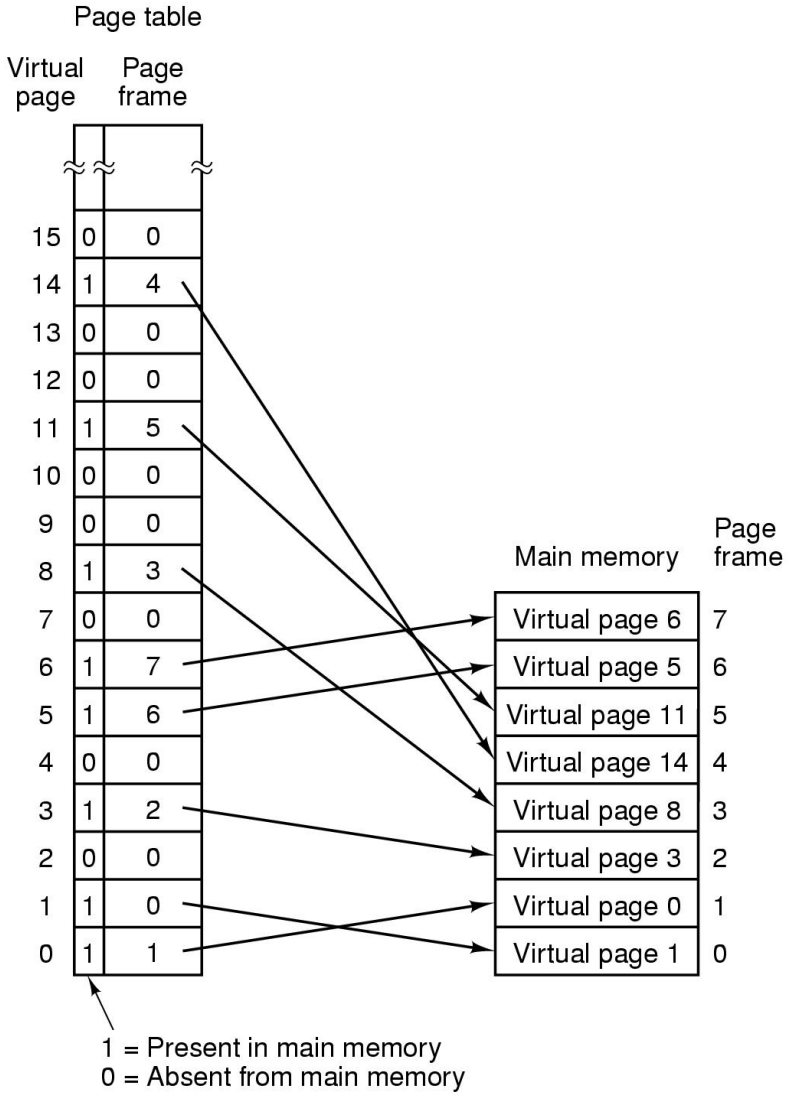
- La crescita delle prestazioni dei calcolatori è stata sempre legata alla crescita delle dimensioni della memoria a disposizione
- Nei primi calcolatori l'unica memoria a disposizione era quella "principale". Obiettivo dei programmatori: compattare i programmi !!!
- Memoria "secondaria": al programmatore l'onere di dividere il programma in parti e gestirne il loro "caricamento" da memoria secondaria a principale.
- Gestione automatica del caricamento in memoria: **memoria virtuale** (1961).
- **Paginazione**: la memoria virtuale e' divisa in pagine e politiche (FIFO, LastRecentlyUsed – dimensione pagina?) gestiscono il caricamento in memoria e scaricamento – in memoria secondaria – .

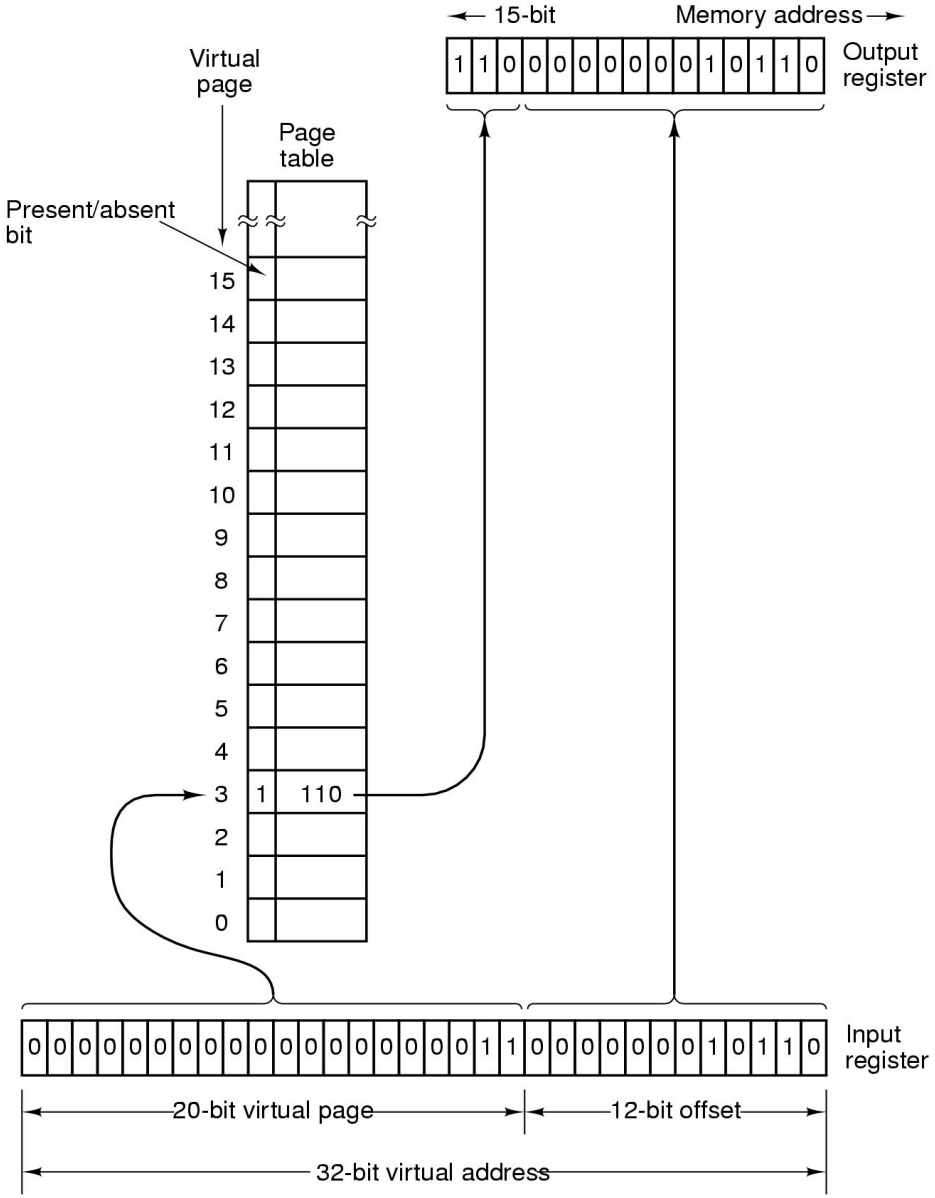
Paginazione



Codifica dell'indirizzo

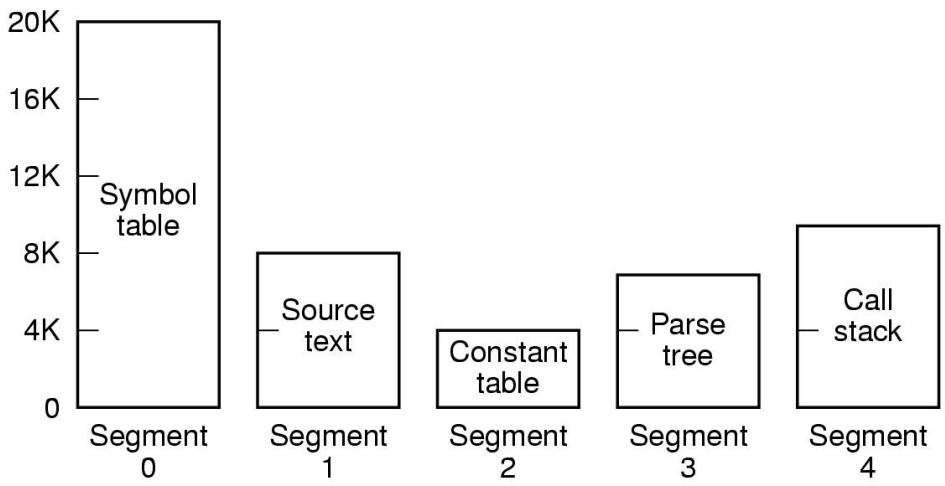
indirizzo virtuale → indirizzo fisico



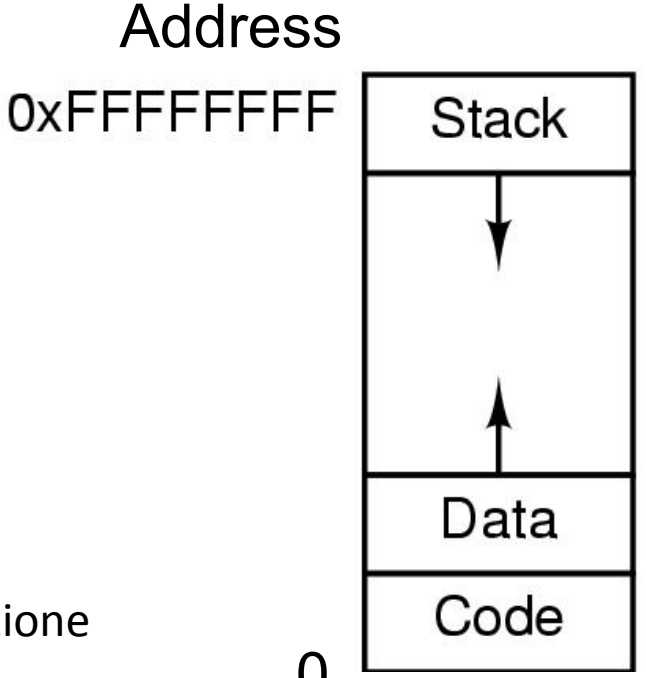


Segmentazione dello spazio di memoria virtuale

- Dividere la memoria virtuale in parti dette Segmenti permette di disporre di spazi di memoria che possono crescere “autonomamente”
- La dimensione di tali spazi è fissata per evitare sovrapposizioni



Organizzazione logica

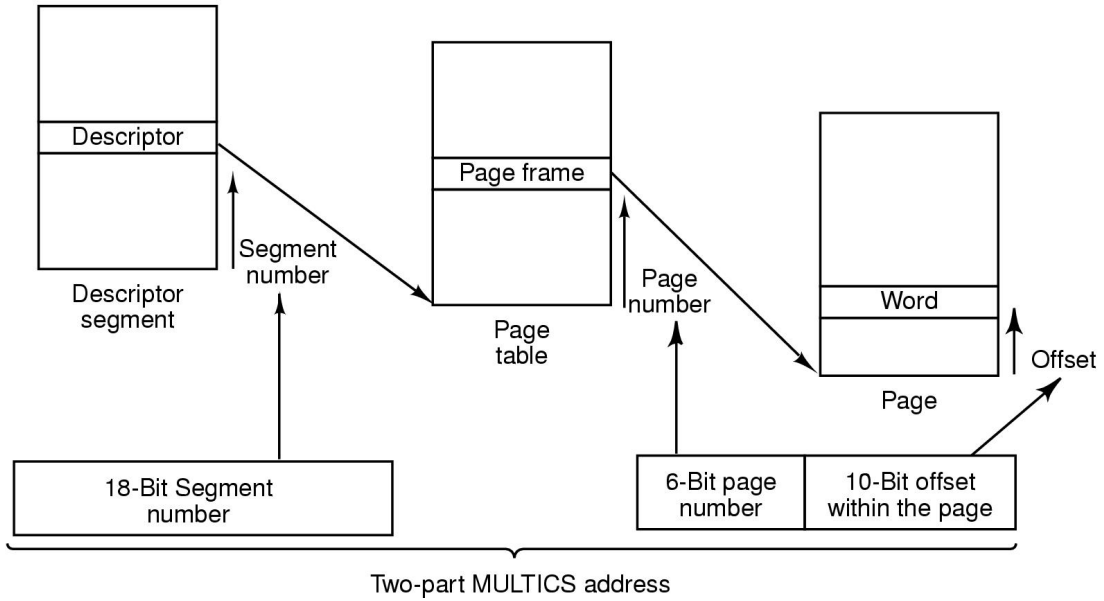


Implementazione fisica

Ricostruzione dell'indirizzo fisico

MULTICS

- MULTiplexed Information and Computing Service (MIT, Bell Labs and GE, 1965)
- MULTICS è uno dei primi sistemi operativi a combinare segmentazione e paginazione.



Windows, UNIX .....*si rimanda al libro di testo paragrafo 6.4*

Organizzazione della memoria in 8088

- Lo spazio di memoria indirizzabile dalla CPU è diviso in segmenti logici: ***Segmentazione della memoria***
  - **CODICE**
  - **DATI ***
  - **STACK**
  - **EXTRA***

  ] SEGMENTI

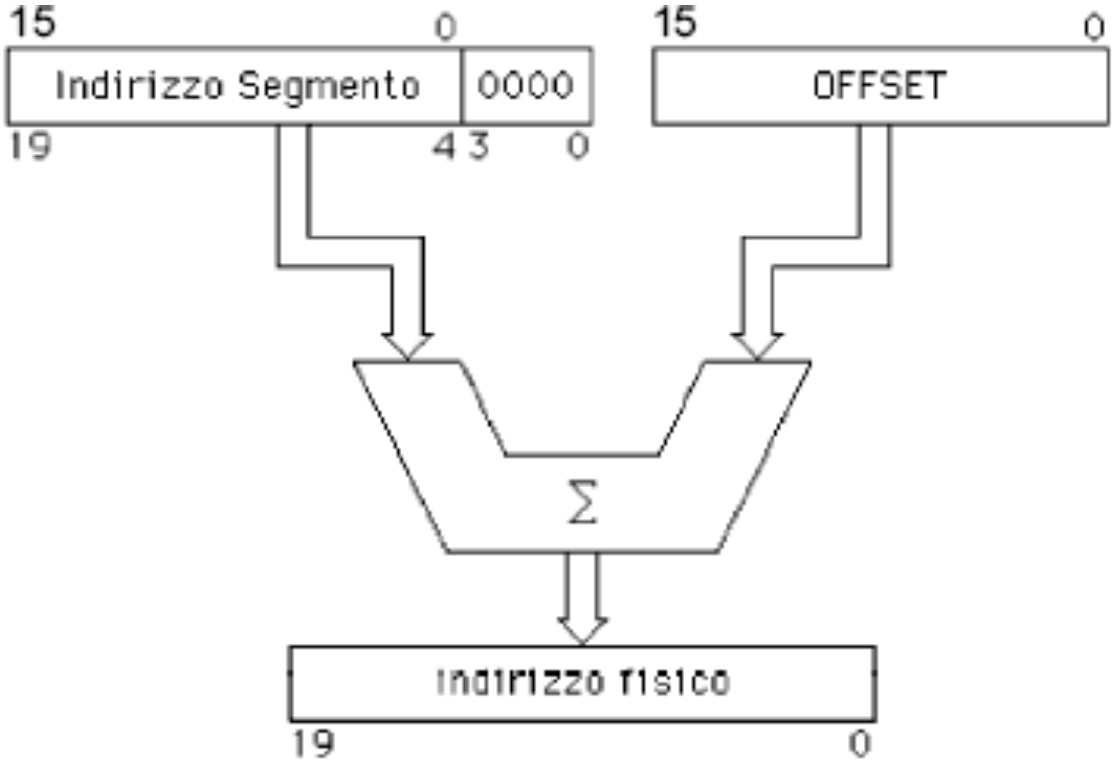
- L'8088 può gestire al massimo 4 segmenti contemporaneamente

Segmenti

OGNI SEGMENTO:

- è un'unità logica di memoria indipendente, indirizzabile separatamente dalle altre unità
- è al massimo di 64k byte
- è costituito da locazioni contigue di memoria e organizzate a **Word** da 16bit
- inizia a un indirizzo di memoria multiplo di 16 (si dice allineamento a 16 byte) ed è quindi identificabile univocamente dai primi 16 bit del suo indirizzo di partenza in memoria

Ricostruzione indirizzo fisico

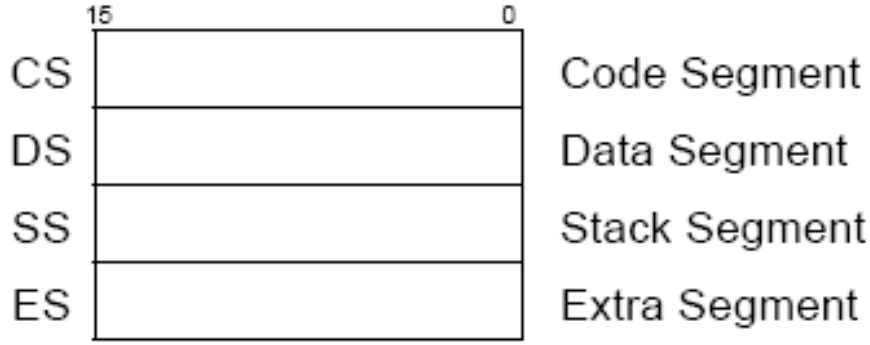


La BIU si occupa di questo !!!

Registri di SEGMENTO

La CPU usa 4 registri per ***puntare*** ai 4 segmenti ***correnti***

- **CS** (Code Segment) punta al segmento codice corrente: il segmento da cui vengono ottenute le istruzioni da eseguire
- **SS** (Stack Segment) punta al segmento contenente lo stack corrente
- **DS** (Data Segment) punta al segmento dati corrente: in genere tale segmento contiene variabili di programma
- **ES** (Extra Segment) punta al segmento extra corrente: in genere anche questo segmento viene utilizzato per memorizzare dati



Vediamo il funzionamento dei singoli segmenti:

SEGMENTO CODICE

SEGMENTO DATI *

SEGMENTO STACK

SEGMENTO EXTRA *

*contengono entrambi dati

SEGMENTO CODICE

- Contiene le istruzioni da eseguire
- Il registro PC punta ad un'istruzione alla volta
- Il registro PC punta alla prossima istruzione da eseguire
- viene gestito dal BIU
- PC è dato da CS:IP cioè CS*16+IP
- CS (registro a 16 bit) punta al segmento codice corrente
- IP (registro a 16 bit) punta alla prossima istruzione da eseguire nel segmento corrente.
- IP contiene, in ogni istante, l'offset (cioè la distanza in byte) dell'istruzione successiva dall'inizio del segmento
- I programmi non hanno accesso diretto all'IP, ma le istruzioni lo modificano implicitamente

PC = CS:IP

ProgramCounter = CodeSegment : InstructionPointer

SEGMENTO STACK

- memoria gestita con politica LIFO
- realizzato in memoria centrale
- Indirizzato dai registri SS e SP
  - SS contiene l'indirizzo del segmento stack
  - SP contiene l'offset del top dello stack
- 64k byte
- se un programma oltrepassa per errore tale limite, ...FATAL ERROR

Cella in cima = SS:SP

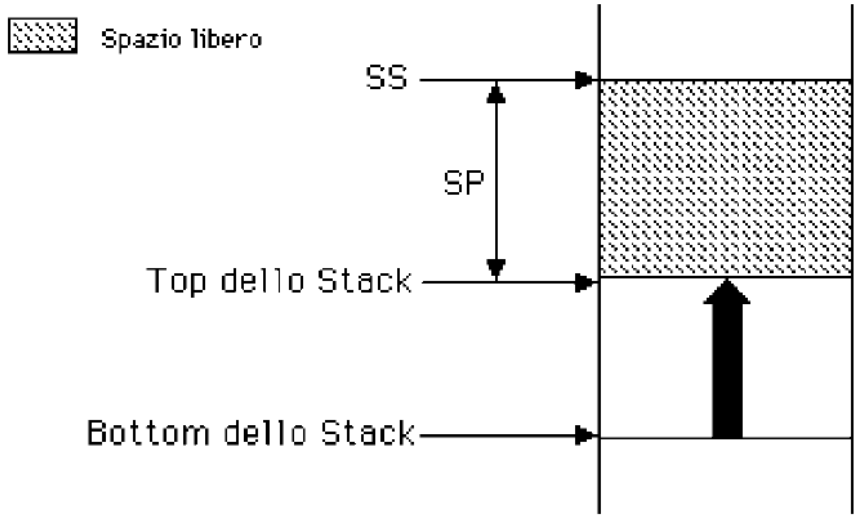
StackSegment : StackPointer

Operazioni di Push e di Pop

- PUSH e POP aggiungono/rimuovono un elemento dalla cima dello stack selezionato da SS:SP.

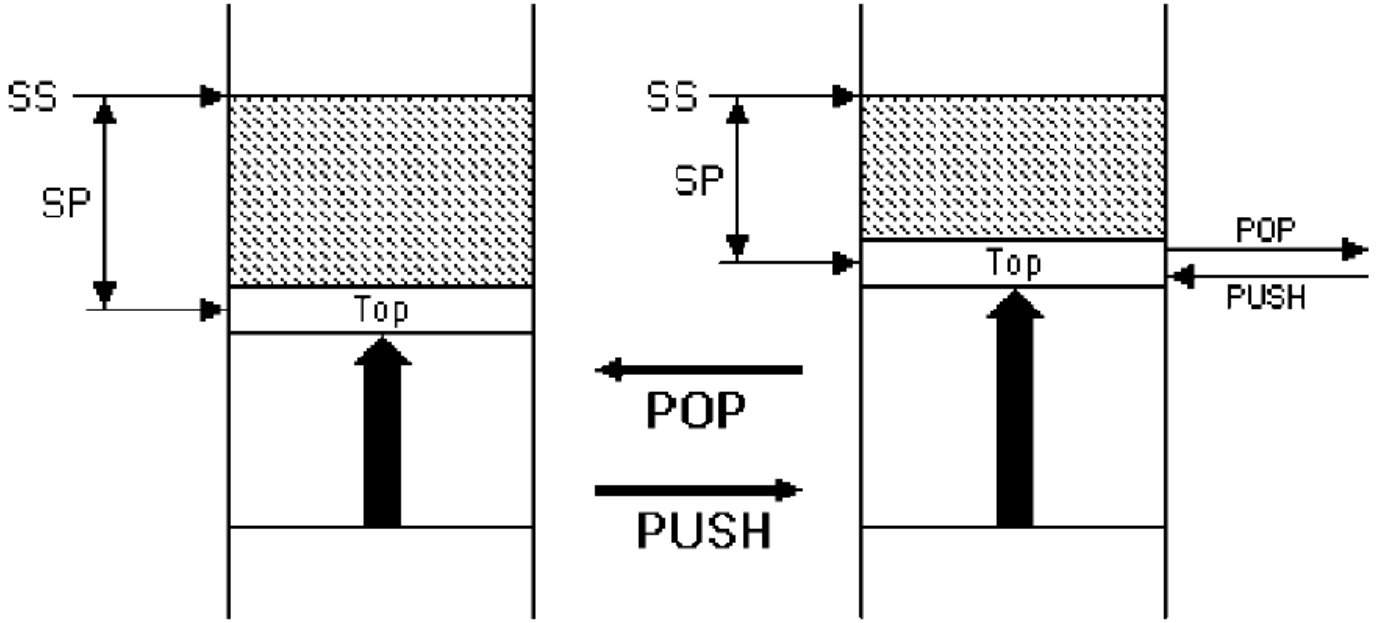
- Le istruzioni **push** e **pop** agiscono sul segmento stack corrente modificando SS e SP

- Le operazioni PUSHF e POPF trasferiscono il contenuto del registro flag nella cima dello stack e viceversa.



- Operazione di **push (SCRITTURA)**:
  SP = SP – 2
  di una word al nuovo top
- Operazione di **pop (LETTURA)**:
  SP = SP + 2
  di una word al top

LIFO



???

Perché

???????

- SP = SP – 2
- SP = SP + 2

PUSH:

- **operando immediato** o **indirizzo effettivo**

```
PUSH 30          operando immediato
PUSH BX          indirizzo effettivo
PUSH [BX+DI]     indirizzo effettivo (con registro)
PUSH [BX+DI+2]   indirizzo effettivo (con reg.+spiazzamento)
```

- è implicito il “dove mettere il dato”, cioè SP è implicito

POP:

- **indirizzo effettivo**

```
POP BX          indirizzo effettivo
POP [BX+DI]     indirizzo effettivo (con registro)
POP [BX+2+DI]   indirizzo effettivo (con reg.+spiazzamento)
```

- è implicito il “da dove prelevare il dato”, cioè SP è implicito

SEGMENTO DATI

- Contiene i dati divisi in dati inizializzati e dati non inizializzati

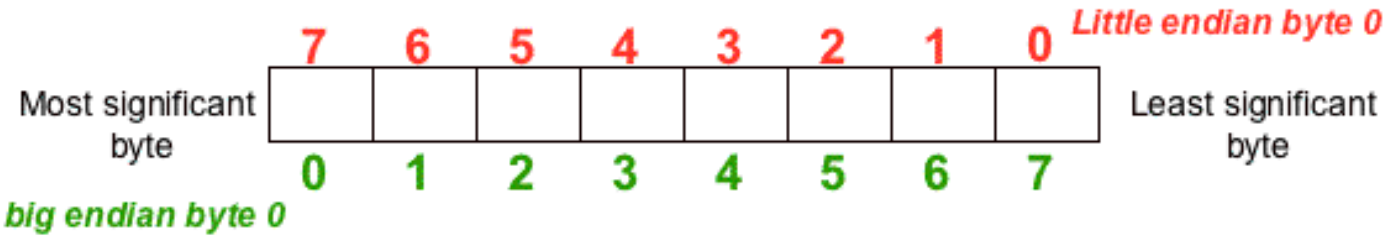
- Tipi di dato:
  - Byte 8 bit
  - Word 16 bit
  - Double (long) 32 bit
    Si usano 2 registri per contenere il Double: DX:AX con AX che contiene la parte meno significativa
  - Decimali binari (Binary Coded Decimal, BCD)
    XX decimale → binario (1 word)

- Ordinamento **Little Endian** (INTEL)

Little endian vs. Big endian

- **Big Endian**: address of most significant byte = word address (big end of the word).
  - 0xDEADBEEF = DE AD BE EF
  - IBM 360, Power; Motorola 68k; MIPS; SPARC; HP PA

- **Little Endian:** address of least significant byte = word address (little end of the word)
  - 0xDEADBEEF = EF BE AD DE
  - Intel 80x86; DEC Vax; DEC Alpha

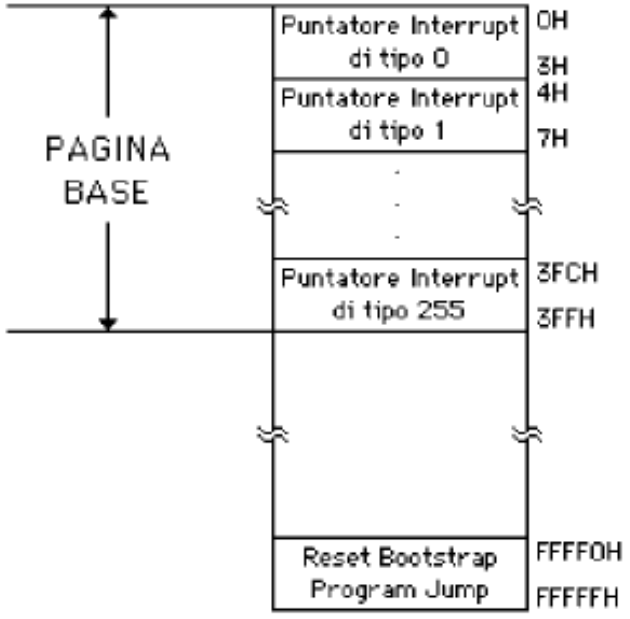


Fonte: http://asci-training.lanl.gov/BProc/

Locazioni di memoria Riservate

- Le locazioni da 0h a 3FFh (1024 byte) sono dedicate al servizio di 256 tipi di interrupt:

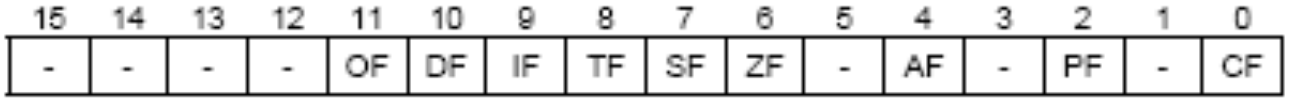
- Le locazioni da 0FFFF0h a 0FFFFFh (16 byte) sono dedicate alla gestione del reset:
  - contengono un salto alla routine da eseguire in caso di reset della CPU;
  - dopo un reset i registri CS e IP vengono inizializzati rispettivamente a 0FFFFh e a 0h



Registro dei FLAG

Registro a 16 bit contenente:

- **6 flag di stato** - modificati dall'EU in base al risultato di operazioni logiche e aritmetiche (es. riporto, overflow, segno, parità)
- **3 flag di controllo** – modificabili da programma al fine di controllare il comportamento della CPU (es. abilitare/disabilitare interrupt, funzionamento passo-passo per debugging)
- Esistono istruzioni che permettono al programma di controllare il contenuto di tali flag a fini decisionali

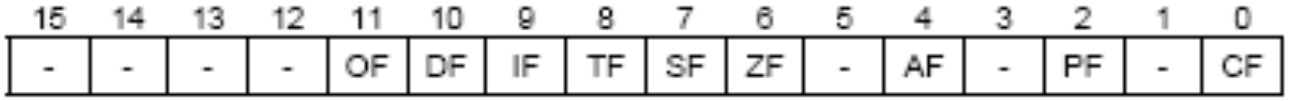


FLAG DI CONTROLLO

- DF (Direction Flag) - indica la **direzione** secondo la quale operare sulle stringhe (da destra a sinistra o viceversa)

- IF (Interrupt-enable Flag) - **abilita** o **disabilita** gli **interrupt esterni** mascherabili (non ha effetto sugli altri tipi di interrupt)

- TF (Trap Flag) - pone il processore nella modalità **single-step** per il debugger. In questa modalità, la CPU genera automaticamente un interrupt interno dopo ogni istruzione, in modo che un programma possa essere controllato istruzione per istruzione



FLAG DI STATO

- AF (Auxiliary Flag) - condizione di **riporto** durante un'operazione **BCD** (Binary Coded Decimal)
- CF (Carry Flag) - condizione di **riporto** durante un'istruzione aritmetica
- OF (Overflow Flag) - **overflow** aritmetico
- SF (Sign Flag) - **segno** del risultato
- PF (Parity Flag) - se il **numero di bit** a 1 del risultato è pari, vale 1, altrimenti, vale 0
- ZF (Zero Flag) - indica che il **risultato** è **0**