ARCHITETTURA DEI MICROPROCESSORI INTEL 8086/8088

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

Caratteristiche	8086/8088	80286	Pentium
address bus	20 bit	24 bit	32 bit
data bus	16 / 8 bit	16 bit	64 bit
Indirizzamento	1M byte	16M byte	4G byte
		1G byte virtual	64T byte virtual
Registri	16 bit	16 bit	32 bit
Piedini	40	68	267

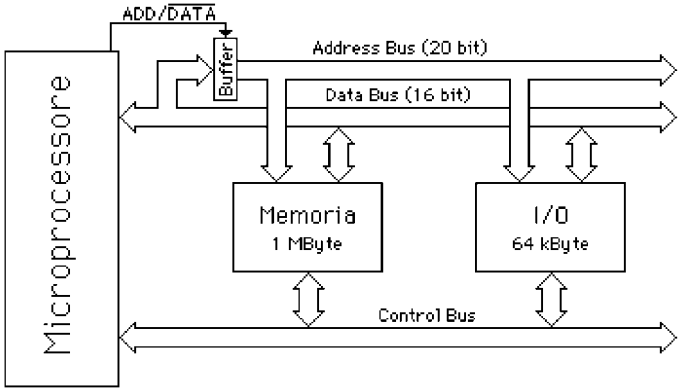
Compatibilità del software (assembler):

Intel

8086/8088 ➡ 80286 ➡ 80386 ➡ 80486 ➔ Pentium ➔ P6. . .

Motorola

68000 ➡ 68020 ➡ 68030 ➡ 68040 ➡ PowerPC . . .



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 FFFFFH

Accesso a 4 blocchi di memoria di 64k byte ciascuno (segmenti)

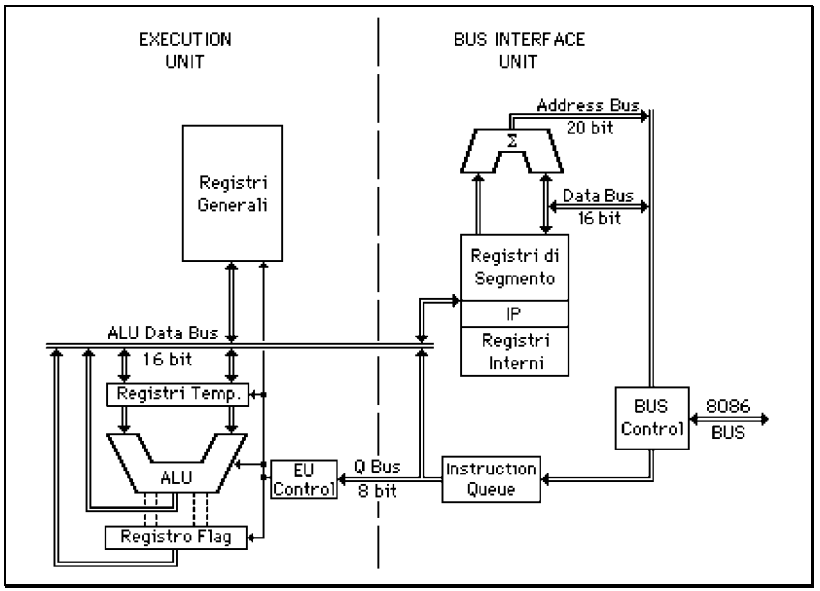
Esecuzione di un programma

- Il programma è caricato in memoria centrale
- Si compone di due parti fondamentali:
  istruzioni (“codice”) e dati
- Il microprocessore inizia la lettura della prima istruzione a un indirizzo noto di memoria;
  una volta letta, esegue l’istruzione
- Il microprocessore legge ed esegue l’istruzione successiva in memoria, e così via
- Alcune istruzioni particolari, dette di trasferimento di controllo (“salti”, chiamate a procedura, interruzioni, ...) modificano arbitrariamente l’indirizzo da cui è letta la successiva istruzione
- Ogni istruzione può o meno fare riferimento a un dato (o più dati) in memoria; in tal caso, viene calcolato l’indirizzo del dato ed eseguita un’operazione di lettura e/o scrittura all’indirizzo di memoria

Il microprocessore o CPU

La CPU è costituita da due blocchi funzionali:

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



Execution Unit

- **esegue le istruzioni**
- **fornisce dati e indirizzi al BIU**
- **modifica registri generali e registro flag**

ALU, registri e bus interno a 16 bit (8086/8088)

L'EU non ha connessioni dirette con il bus di sistema (cioè, con il mondo esterno)

Quando l'EU deve eseguire una nuova istruzione, la ottiene dalla coda gestita dal BIU e se la coda è vuota si pone in attesa
Quando un'istruzione richiede di accedere alla memoria o a un device periferico, l'EU richiede al BIU di ottenere o memorizzare il dato

Gli indirizzi manipolati dall'EU sono di 16 bit
Il BIU effettua le operazioni che permettono di accedere all'intero spazio di memoria disponibile

Bus Interface Unit

Il BIU esegue tutte le richieste dell'EU che coinvolgono il mondo esterno (e quindi il bus di sistema), cioè i trasferimenti di dati tra la CPU e la memoria o i dispositivi di I/O

- **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)**

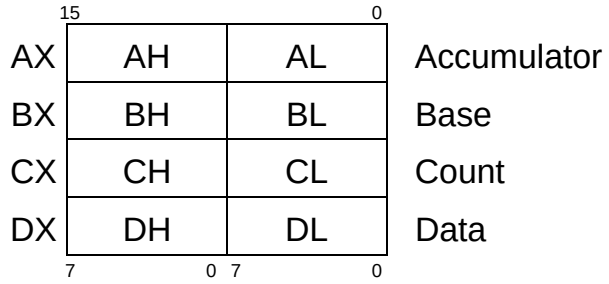
Le istruzioni caricate dal BIU nella coda sono quelle che *seguono* l'istruzione correntemente in esecuzione nell'EU

Se l'EU esegue un'istruzione di salto, il BIU svuota la coda e comincia a riempirla di nuovo a partire dal nuovo indirizzo
in questo caso, l'EU deve aspettare che la BIU abbia acquisito la nuova istruzione da eseguire

Registri Generali

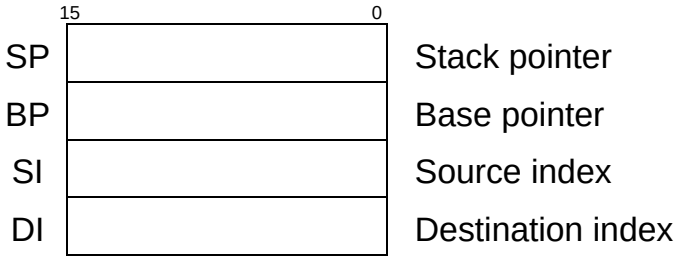
Data register (AX, BX, CX, DX)
Pointer register (SP, BP)
Index register (DI, SI)

Data Register



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

Pointer e Index Register



Mancanza di Ortogonalità

Ortogonalità: possibilità per le istruzioni di utilizzare uno qualsiasi dei registri come operando

L'8086 manca di ortogonalità

Esistono:

istruzioni che utilizzano i registri generali in modo implicito

ad esempio: le istruzioni che agiscono sullo stack coinvolgono sempre, in modo implicito, il registro SP

istruzioni che funzionano solo con particolari registri generali

ad esempio: l'istruzione per la lettura (scrittura) di un byte da una porta di I/O ha sempre il formato:

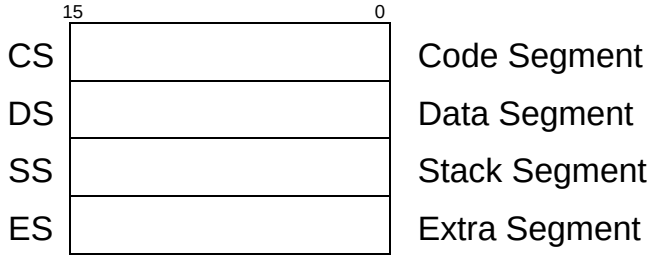
IN AL, #porta *(OUT #porta, AL)*

Registri di Segmento

Lo spazio di memoria indirizzabile dalla CPU è diviso in segmenti logici (massimo di 64k byte)

La CPU può accedere direttamente a 4 segmenti per volta (massimo 256k byte)

Quattro registri di segmento puntano ai quattro segmenti correntemente *attivi* :



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

Segmentazione fisica della memoria

Lo spazio di memoria viene visto come un gruppo di segmenti

Ogni segmento:

- è un'unità logica di memoria indipendente, indirizzabile separatamente dalle altre unità
- inizia a un indirizzo di memoria multiplo di 16
- è costituito da locazioni contigue di memoria
- è al massimo di 64k byte

I segmenti si dicono allineati ai 16 byte (o allineati al paragrafo)

Ogni segmento è identificabile univocamente dai 16 bit più significativi del suo indirizzo di partenza in memoria:

indirizzo fisico 220h

indirizzo segmento 22h

I quattro registri segmento puntano ai quattro segmenti correntemente utilizzabili

Ogni programma in esecuzione può accedere direttamente a:

64k byte di codice - CS

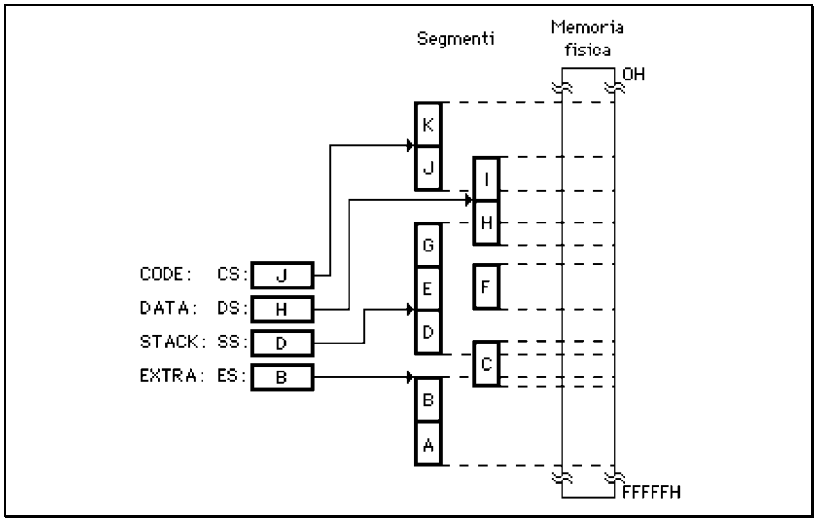
64k byte di stack - SS

128k byte di dati - DS e ES

Per accedere al codice o ai dati contenuti in altri segmenti, è necessario modificare i registri segmento in modo opportuno

I segmenti possono essere comunque disposti in memoria; ad esempio:

- **contigui** (segmenti A e B)
- **disgiunti** (segmenti A e C)
- **sovrapposti parzialmente** (segmenti B e C)
- **sovrapposti totalmente** (segmenti E e F)

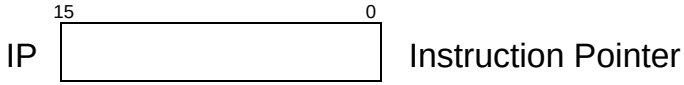


Una locazione della memoria fisica può essere contenuta in più segmenti

Instruction Pointer

- registro a 16 bit IP
- viene gestito dal BIU
- contiene, in ogni istante, l'offset (cioè la distanza in byte) dell'istruzione successiva dall'inizio del segmento codice corrente (CS)

I programmi non hanno accesso diretto all'IP, ma le istruzioni lo modificano implicitamente



Il program counter classico coincide con CS:IP.

Registro Flag

Registro a 16 bit contenente:

6 flag di stato

3 flag di controllo

I rimanenti bit non sono utilizzati

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF

I flag di stato vengono modificati dall'EU in base al risultato delle operazioni logiche e aritmetiche

Esiste un gruppo di istruzioni che permette al programma di controllare il contenuto di tali flag a fini decisionali

I flag di controllo possono essere settati o azzerati dal programma al fine di modificare il comportamento della CPU

Generazione dell'indirizzo fisico

Un indirizzo fisico è un valore di 20 bit che identifica in modo univoco ogni byte dello spazio di memoria di 1M byte

Per trasferire dati tra la CPU e la memoria è necessario utilizzare gli indirizzi fisici

I programmi utilizzano indirizzi formati da:

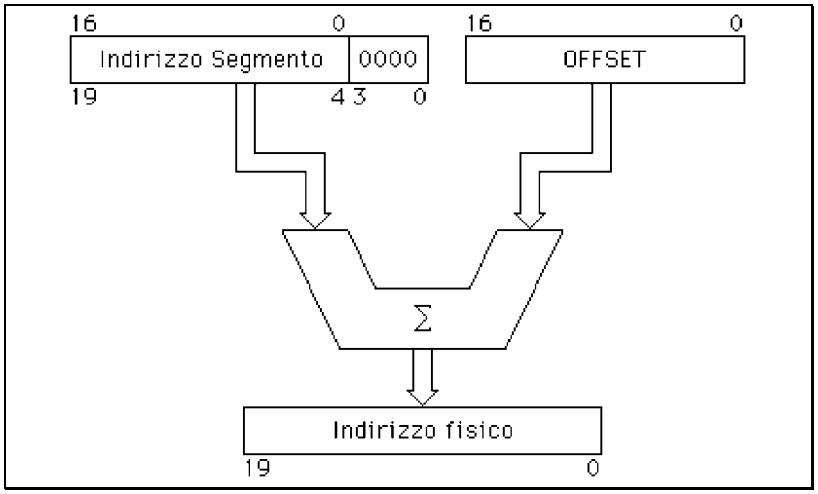
- indirizzo del segmento
- offset nel segmento

entrambi quantità di 16 bit senza segno

segmento : offset

Il BIU converte la coppia segmento:offset in indirizzo fisico

Ciò avviene moltiplicando l'indirizzo del segmento per 16 e sommando al risultato l'offset nel segmento



Lo stesso indirizzo fisico può essere ottenuto con diverse coppie segmento:offset

L'indirizzo fisico 1000H può essere ottenuto:
con 100H:0H
con F0H:100H
con E0H:200H
etc...

Il BIU ottiene la coppia segmento:offset da traslare, in modi diversi

Le istruzioni da eseguire vengono ricavate dal segmento codice corrente, pertanto l'indirizzo fisico della successiva istruzione è dato da: CS:IP, cioè CS*16+IP

Le istruzioni che agiscono sullo stack utilizzano il segmento stack corrente:
- SS contiene l'indirizzo del segmento
- SP contiene l'offset del top dello stack

Gli operandi che fanno riferimento alla memoria (variabili di programma) di norma risiedono sul data segment corrente (DS)
però, il programma può dire al BIU di utilizzare uno qualunque dei quattro segmenti correntemente disponibili
L'offset della variabile viene invece calcolato dall'EU e dipende dalla modalità di indirizzamento specificata nell'istruzione

Stack

• area di memoria gestita LIFO (ad esempio, può contenere i record di attivazione delle procedure)

• realizzato in memoria centrale

• definito dai registri SS e SP

In memoria possono coesistere più stack, ognuno al massimo di 64k byte
se un programma oltrepassa per errore tale limite, ...

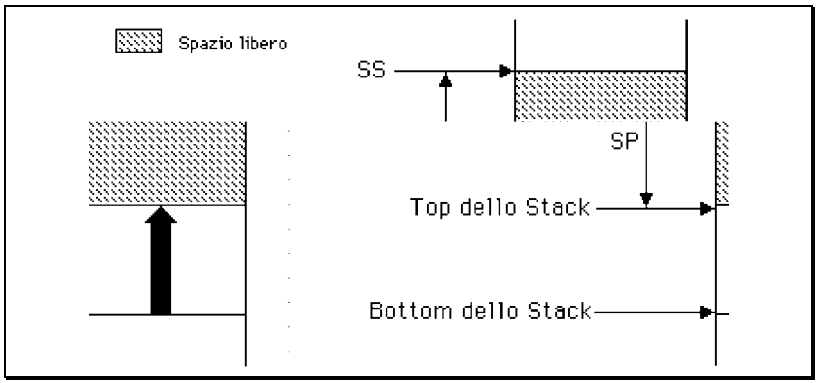
Un solo stack è quello corrente:

• SS contiene l'indirizzo del segmento stack

• SP contiene l'offset del top dello stack

Lo stack cresce andando dagli indirizzi alti a quelli bassi:

l'indirizzo di partenza dello stack (contenuto in SS) non è il bottom dello stack



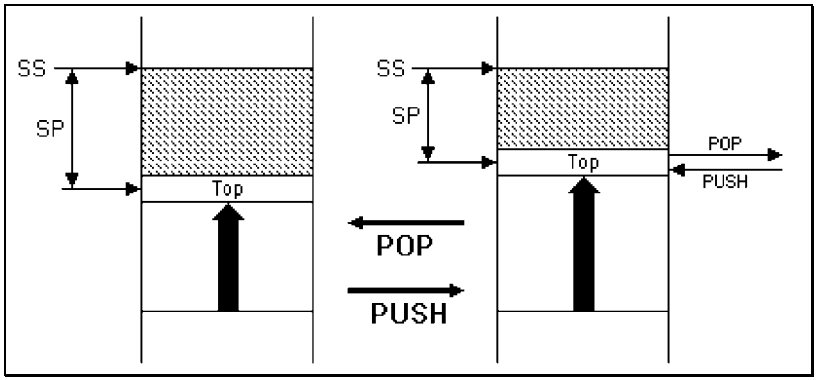
Le istruzioni che operano sullo stack trasferiscono due byte per volta (una word)

Operazione di **push**:

- SP ➔ SP - 2
- scrittura di una word al nuovo top

Operazione di **pop**:

- lettura di una word dal top
- SP ➔ SP + 2



Interrupt

Un interrupt (interruzione) è un evento che si verifica in momenti non prevedibili. L'effetto è quello di trasferire il controllo a una nuova locazione di memoria, in cui è collocata una procedura, detta routine di servizio dell'interrupt. Al termine della procedura, si rientra nel programma principale.

Gli interrupt possono essere:

- hardware
- software

Gli **interrupt hardware**:

- hanno origine dalla logica esterna
- permettono di gestire eventi asincroni
- si dividono in:
  - mascherabili
  - non mascherabili

Gli **interrupt software**:

- hanno origine dall'esecuzione del programma:
  - **direttamente**
    (per es., l'esecuzione di un'istruzione INT)
  - **indirettamente** - condizioni eccezionali
    (per es., una divisione per zero)

Gli interrupt nell'8086

Le locazioni da 0H a 3FFH contengono una tabella (Interrupt Vector Table) con 256 ingressi

Ogni ingresso contiene due valori di 16 bit che forniscono l'indirizzo della routine di servizio dell'interrupt e che vengono caricati nei registri CS e IP quando l'interrupt viene accettato

I primi cinque elementi della tabella sono dedicati a particolari tipi di interrupt predefiniti nell'8086

I successivi 27 elementi sono riservati all'hardware del sistema di elaborazione e non devono essere utilizzati

I rimanenti elementi (da 32 a 255) sono disponibili per le routine di servizio e del sistema operativo dell'utente

Un programma può **anche** generare eslpicitamente un interrupt di tipo n, mediante l'istruzione **INT n**

- Netta separazione di ambienti tra il programma e la procedura
- Trasferimento del controllo a routine attraverso indirizzi non noti al programma

Esempi:

Interrupt 0 (Divide Error) - segnala un errore durante un operazione di divisione (ad es., divisione per zero)

Interrupt 1 (Single Step) - un'istruzione dopo il set-taggio di TF (permette di eseguire una singola istruzione all'interno di un programma - utilizzato dal debugger)

Interrupt 2 (Non-Maskable Interrupt) - è l'interrupt hardware di priorità più alta e non è mascherabile - di norma, è riservato ad eventi importanti e urgenti (ad es., una caduta di tensione, un errore nella memoria, un errore sul bus di sistema)

Interrupt 3 (One Byte Interrupt) - utilizzato dal debugger per i breakpoint

Interrupt 4 (Interrupt on Overflow) - condizione di overflow (OF = 1) e viene eseguita l'istruzione INTO; permette di gestire l'eventuale condizione di overflow

Servizio di un interrupt

A livello hardware:

- viene eseguita un push dei registri flags, CS e IP per salvare la situazione corrente e poterla ripristinare al termine del servizio
- vengono caricati i nuovi valori di CS e IP dalla tabella degli interrupt
- vengono azzerati i flag TF (trap per single step) e IF (interrupt)

L'azzeramento di IF disabilita il riconoscimento di ulteriori interrupt hardware nella routine di servizio a meno che tale riconoscimento non venga riabilitato esplicitamente all'interno della routine di servizio stessa

La routine di servizio deve terminare con un'istruzione IRET (Interrupt RETurn), al fine di rispristinare correttamente la situazione presente al momento in cui si è verificata l'interruzione