

Linguaggio Assembler Intel

- cenni -

Calcolatori Elettronici B

a.a. 2005/2006

Massimiliano Giacomini

Un po' di storia

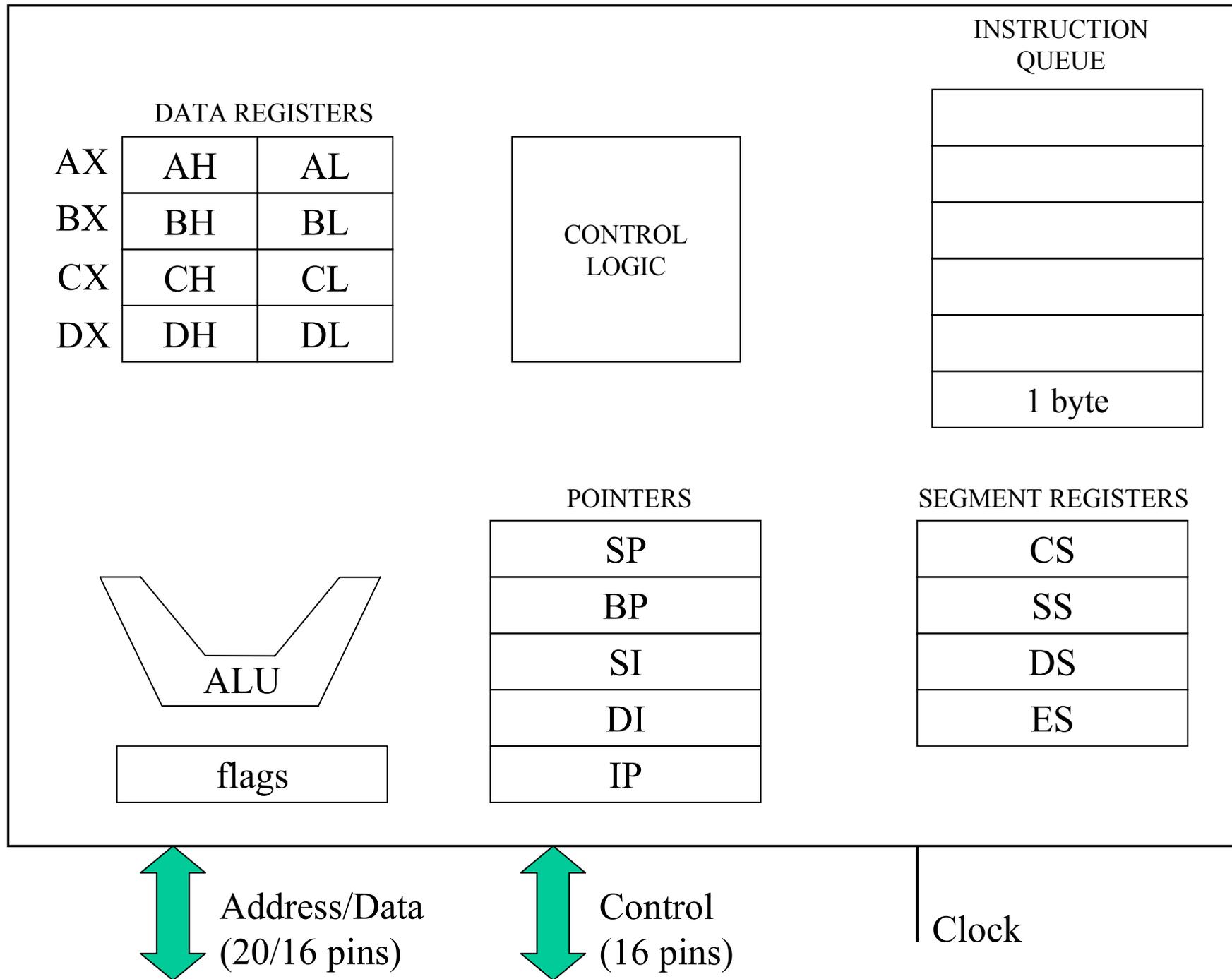
- **1978:** architettura 8086, a 16 bit, con registri dati a 16 bit e spazio di indirizzamento a 20 bit (1 MB). Estende l'8080 (processore a 8 bit). I registri sono dedicati ad usi specifici.
- **1980:** coprocessore matematico 8087 che estende l'8086 con istruzioni in virgola mobile.
- **1982:** architettura 80286, spazio di indirizzamento a 24 bit (16 MB) e aggiunta di istruzioni per memory-mapping & protection (gestione della memoria).
- **1985:** architettura 80386, a 32 bit (registri e spazio di indirizzamento a 32 bit - 4 GB), nuovi modi di indirizzamento e nuove istruzioni. Supporto a paginazione degli indirizzi (come abbiamo visto a proposito della memoria virtuale).
- **1989-1995:** 80486, Pentium, Pentium Pro: miglioramenti per ottenere prestazioni più elevate. Sono aggiunte solo 4 nuove istruzioni al set visibile all'utente per la gestione multiprocessore e il trasferimento dati condizionato.
- **1997-2001:** Pentium II: estensione dell'architettura con set di istruzioni MMX per accelerare le applicazioni legate alla multimedialità e alle comunicazioni; Pentium III: nuovo set di istruzioni Internet SSE per accelerare applicazioni legate a Internet; Pentium IV: istruzioni SSE2.
- **2003-2004:** AMD 64 e Intel "Extended Memory 64 Technology" estendono registri e spazio di indirizzamento a 64 bit. Il processore può comunque funzionare in un apposito modo per garantire la compatibilità con IA-32.

Un primo commento

- Risultato del lavoro indipendente di molti gruppi, aggiunta progressiva al set di istruzioni originali
- Esigenza: mantenere compatibilità all'indietro: vecchi programmi per 8086 dovevano poter funzionare sulle architetture successive
- Ciò ha comportato una complessità crescente dell'architettura
- Nonostante ciò, la famiglia INTEL è la più diffusa:
 - 8086 precede di due anni i “concorrenti” a 16 bit (es. MC 68000)
 - 8088: versione dell'8086 con data-bus a 8 bit per mantenere compatibilità con HW precedente e ridurre i costi
 - 8088 scelto da IBM per PC IBM [MC 6800 usato nel MAC]
 - Diffusione PC IBM (architettura aperta ai “cloni”)
 - Aumento delle risorse economiche da investire per sopperire al problema della complessità crescente

Premessa

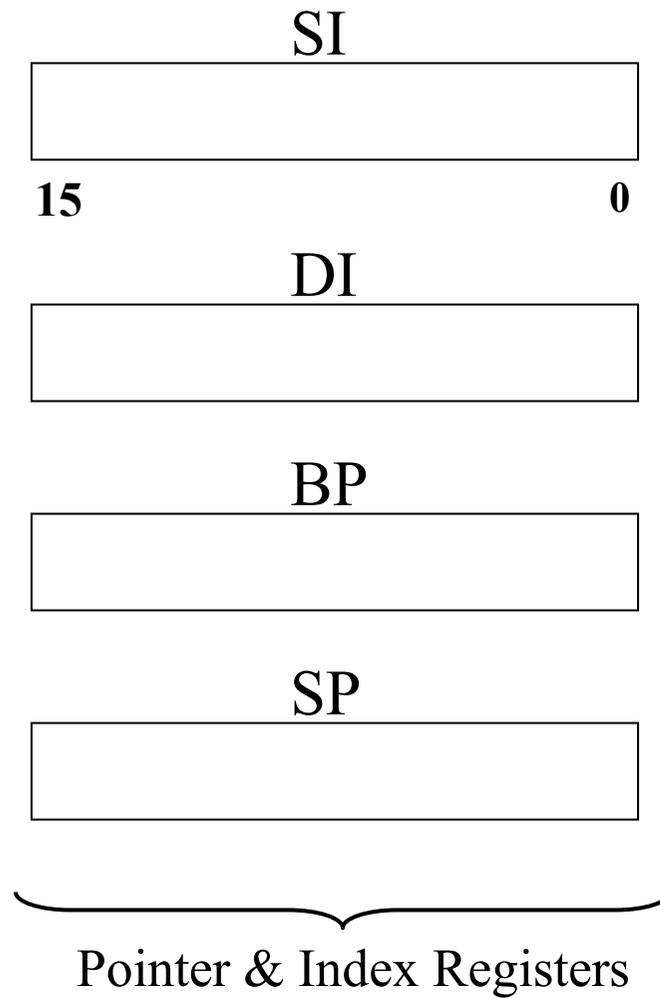
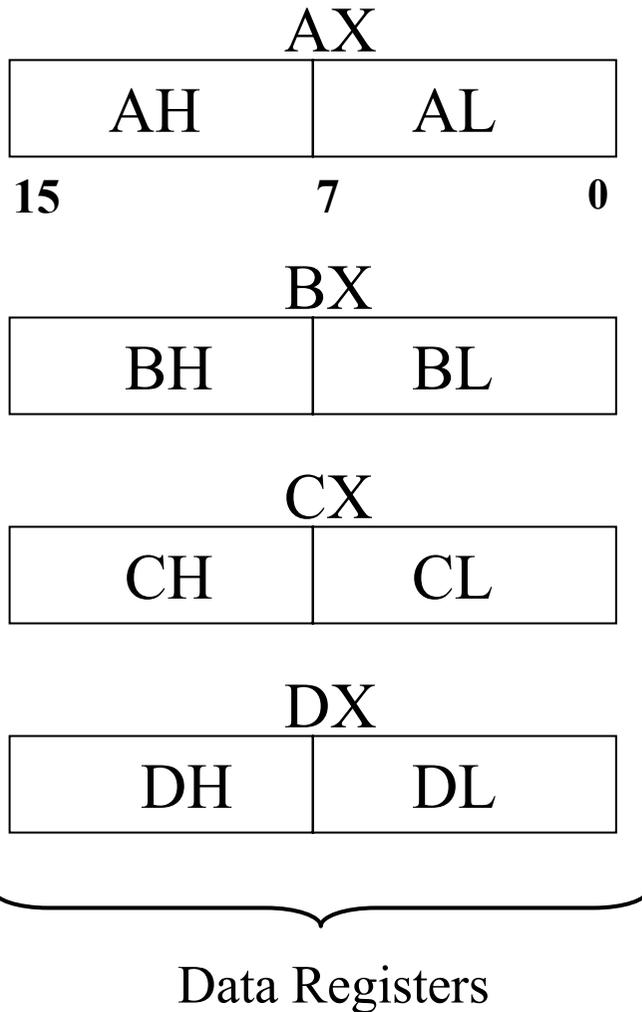
- Faremo riferimento all'architettura 8086 perché offre il set di istruzioni di base che intendiamo esaminare e che sono comuni alle architetture successive
- Quando serve, evidenzieremo le estensioni apportate nell'architettura a 32 bit nata con l'80386



Registri dell'Intel 8086

- **Registri di 16 bit**
- **General-purpose registers:**
 - **ax** (*accumulator*), **bx** (*base*), **cx** (*count*), **dx** (*data*)
 - **si** (*source index*), **di** (*destination index*), **bp** (*base pointer*), **sp** (*stack pointer*)-
- **Segment registers**
 - **cs** (*code segment*), **ds** (*data segment*), **es** (*extra segment*), **ss** (*stack segment*)
- **Special-purpose registers**
 - **ip** (*instruction pointer*)
 - **flags**

Registri “general-purpose” dell’Intel 8086



Nell'8086, ruoli diversi per AX-DX vs. registri puntatori e indice

AX, BX, CX e DX sono registri che servono per memorizzare operandi e risultati di operazioni, fungono da registri aritmetici;

a differenza degli altri, BX può essere usato come “base” nelle modalità di indirizzamento che prevede un “registro base”

Es. Add AX, [BX] è lecito ma non
Add AX, [CX]

CX può essere usato come “contatore implicito” nelle istruzioni di loop

- SP, BP, SI, DI possono essere usati per memorizzare operandi (senza però la possibilità di accedere al byte) ma hanno lo scopo principale di accedere alla memoria:

SP: Puntatore allo stack [usato nelle istruzioni PUSH e POP]

BP: Base Pointer, Registro base per accedere allo stack.

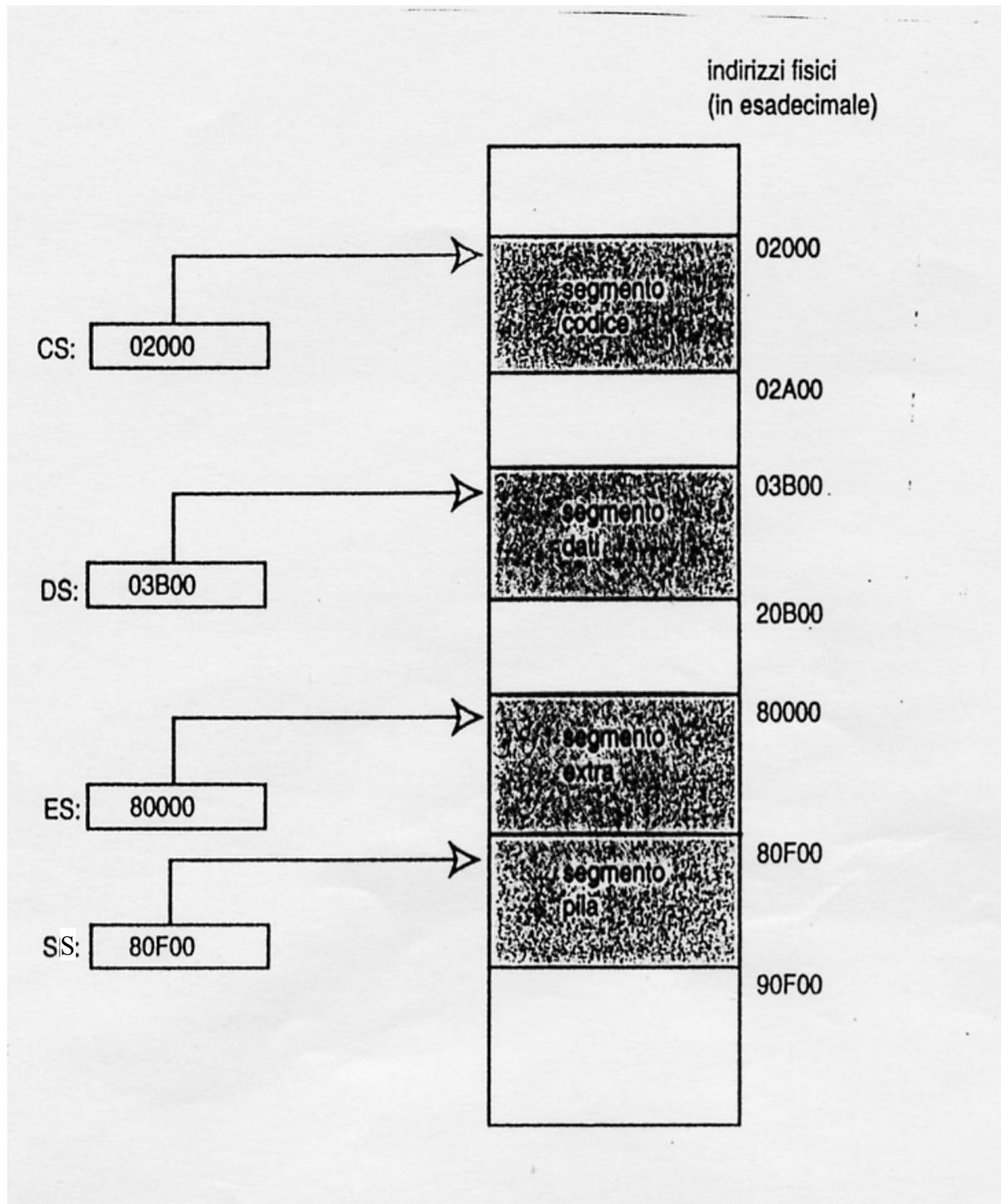
Usato con altri registri indice e/o scostamento.

SI, DI: Registri indice (ma possono essere usati anche come Registri Base).

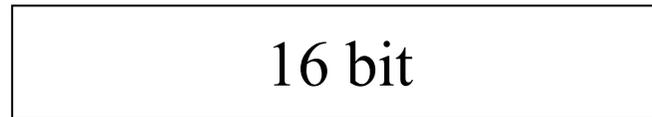
P.es. (BP)+(SI)

I registri segmento

- Permettono di indirizzare 1 MB (2^{20} byte) con registri a 16 bit.
- Lo spazio degli indirizzi assegnato a un programma viene organizzato come un gruppo di aree specializzate chiamate *segmenti*
- Ogni segmento costituisce un'area contigua di memoria
- I registri CS e SS sono usati per indirizzare rispettivamente il *segmento codice* e il *segmento stack*
- Il registro DS è usato per indirizzare il *segmento dati*
- Il registro ES è usato per indirizzare altri segmenti (così come FS e GS aggiunti nelle architetture successive)



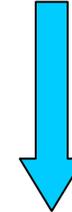
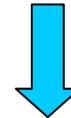
Effective address (EA)



Indirizzo di segmento



Indirizzo fisico



Memoria nell'8086

- La memoria è logicamente organizzata in segmenti
- Un indirizzo di memoria è dato dall'indirizzo del segmento e da un offset all'interno del segmento (*segment:offset*)
- Ad esempio, la CPU preleva sempre le istruzioni dall'indirizzo dato da CS:IP
- Con 16 bit di indirizzamento per l'offset ogni segmento dell'8086 è al massimo di 64K
- Ciò influisce anche sulla programmazione ad alto livello (ad esempio, in certi linguaggi non si possono allocare più di 64K per un array)
- Nei processori dall'80386 in poi un offset può essere sia di 16 che di 32 bit quindi i segmenti possono essere ampi fino a 4 Gbyte
- La memoria viene *indirizzata al byte*
- Una **parola** è formata da 2 byte
- Una **doppia parola** è formata da 4 byte

Calcolo dell'indirizzo fisico nell'8086

- L'indirizzo fisico da porre sul bus degli indirizzi deve essere di 20 bit
- L'indirizzo di segmento, che è di 16 bit, viene moltiplicato per 16, ovvero vengono aggiunti a destra 4 extra bit con valore 0
- All'indirizzo così ottenuto, che è l'inizio dell'indirizzo fisico del segmento viene aggiunto lo scostamento (offset, detto anche *effective address*) di 16 bit

- **Esempio:**

supponiamo [CS] = 123A (esadecimale)

[IP] = 341B (esadecimale)

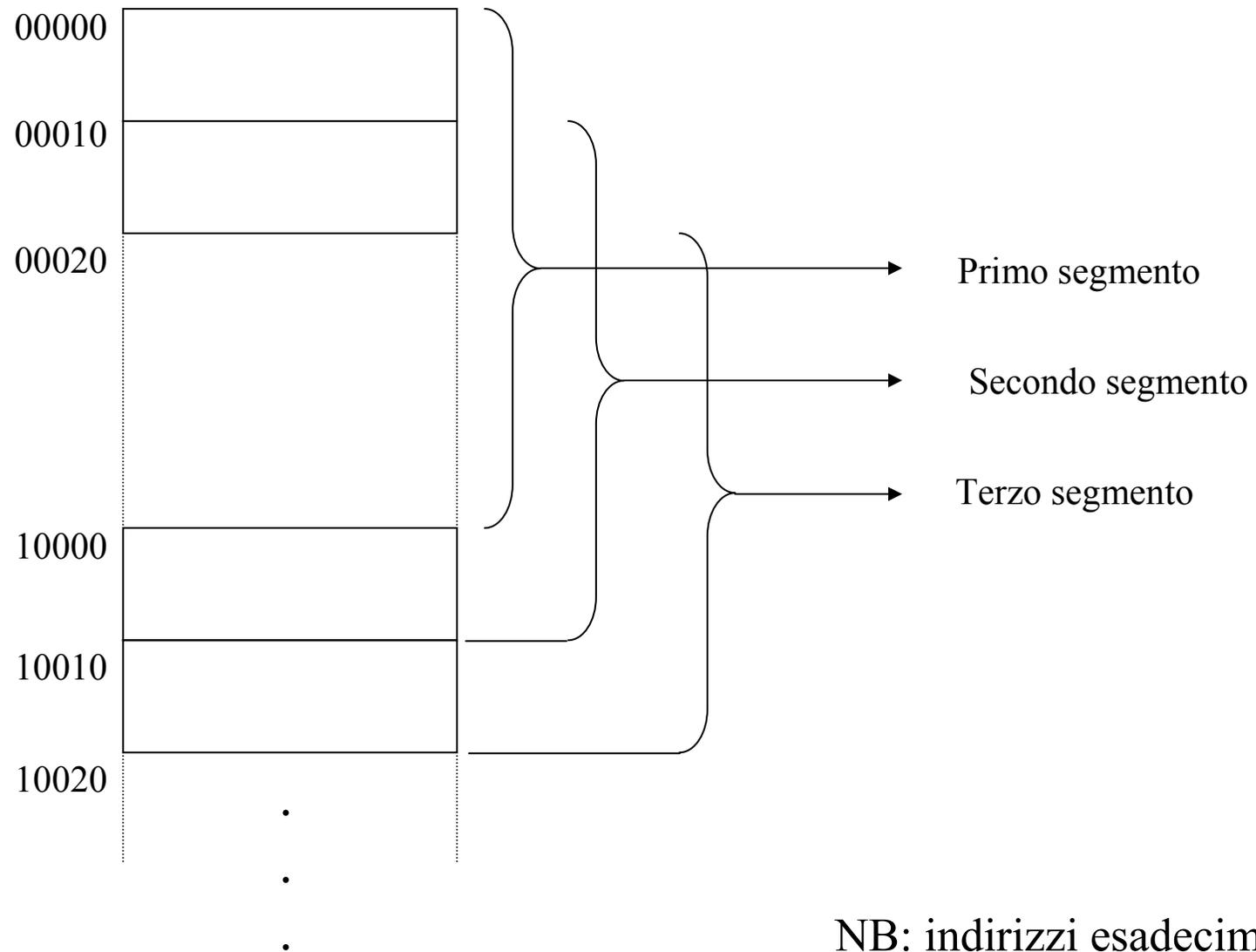
la prossima istruzione sarà prelevata dalla posizione

341B + offset o effective address

123A0 = indirizzo di inizio segmento

157BB indirizzo fisico

Segmentazione della memoria

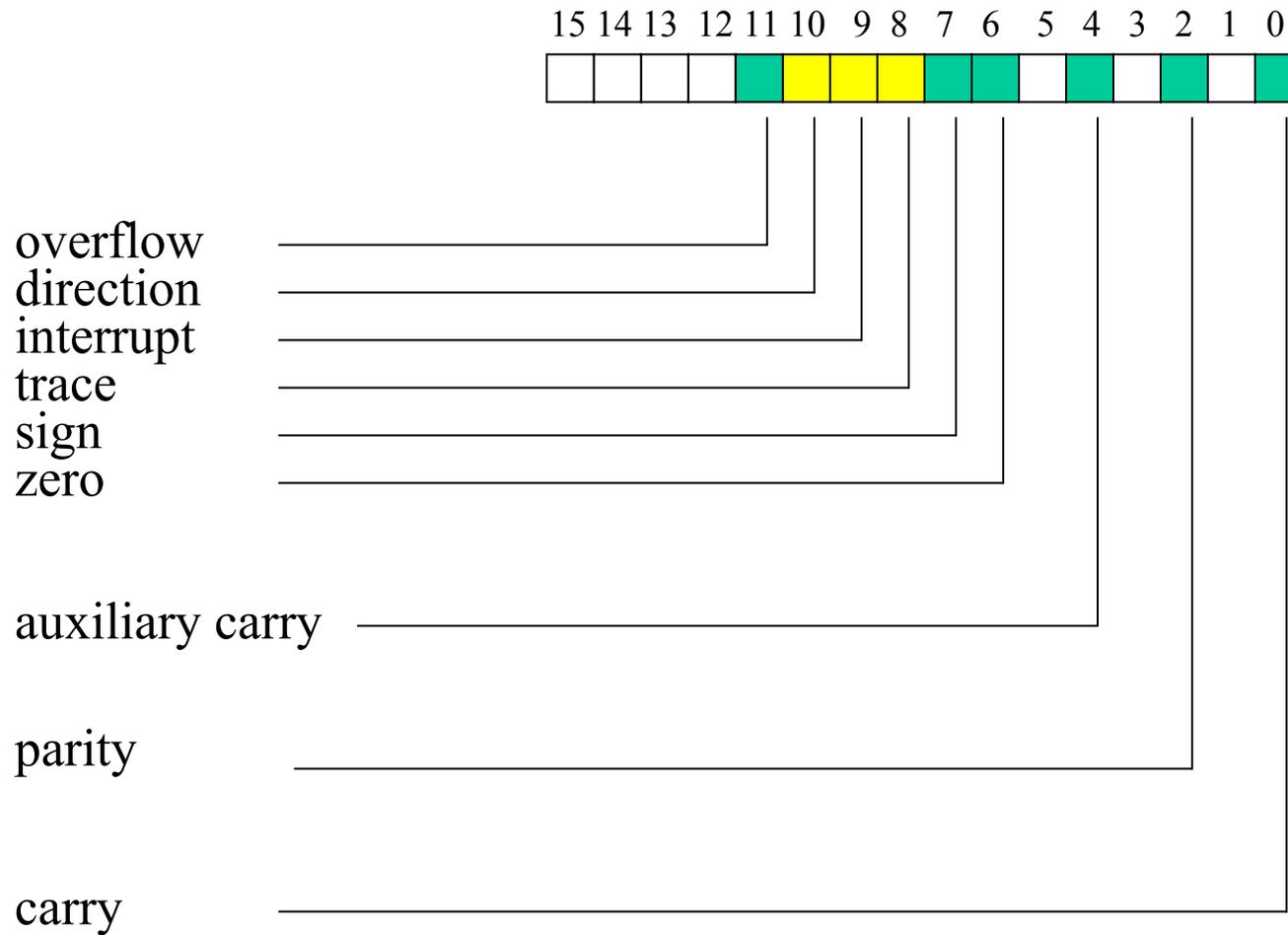


NB: indirizzi esadecimali

Il registro flags

- E' un *registro di stato* in cui sono raggruppati alcuni bit (flags) che fungono da indicatori impostati dalle istruzioni aritmetico-logiche
 - Sono in genere letti da istruzioni di salto condizionato
 - Tra questi:
 - OF**: indicatore di *overflow*. E' posto a 1 quando il risultato di una addizione o sottrazione con segno dà luogo a overflow
 - SF**: indicatore di *segno*. E' posto a 1 quando il risultato di una operazione logico-aritmetica è un numero negativo (= MSB del risultato)
 - ZF**: indicatore di *zero*. E' posto a 1 quando il risultato di una operazione logico-aritmetica vale 0
 - CF**: indicatore di *riporto*. E' posto a 1 quando si genera un riporto o prestito in una istruzione aritmetica (indica l'overflow nel caso di numeri senza segno)
- NB: ADD e SUB operano allo stesso modo per signed e unsigned.
OF è determinato in base alle regole del complemento a 2.

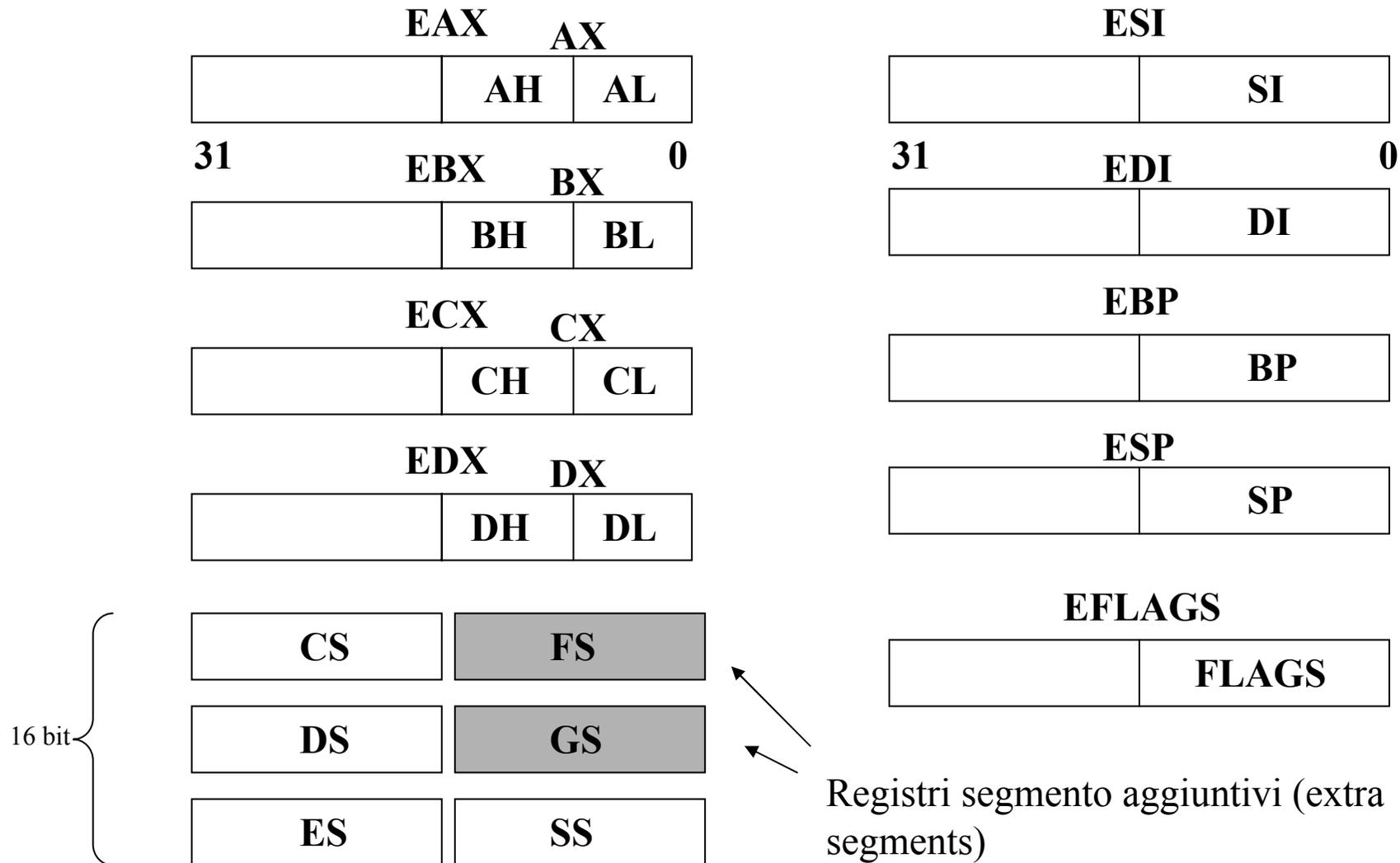
Il registro flags dell'8086



Registri 80386/80486

- Dal punto di vista del programmatore il più importante cambiamento è l'introduzione di **registri a 32 bit**
- I nuovi registri a 32 bit (che ci interessano) sono **eax, ebx, ecx, edx, esi, edi, ebp, esp, eflags, eip**
- Le precedenti versioni a 16 bit rimangono comunque disponibili
- Due nuovi registri di segmento a 16 bit: **fs e gs**
- I registri di segmento sono rimasti a 16 bit
NB: di fatto, l'uso dei registri segmento è in disuso (la capacità di indirizzamento è sufficiente) ma viene assicurata la compatibilità all'indietro
- Aggiunti altri registri usati dal sistema operativo per la gestione della memoria e dei processi, altri registri usati dai programmi di debugging

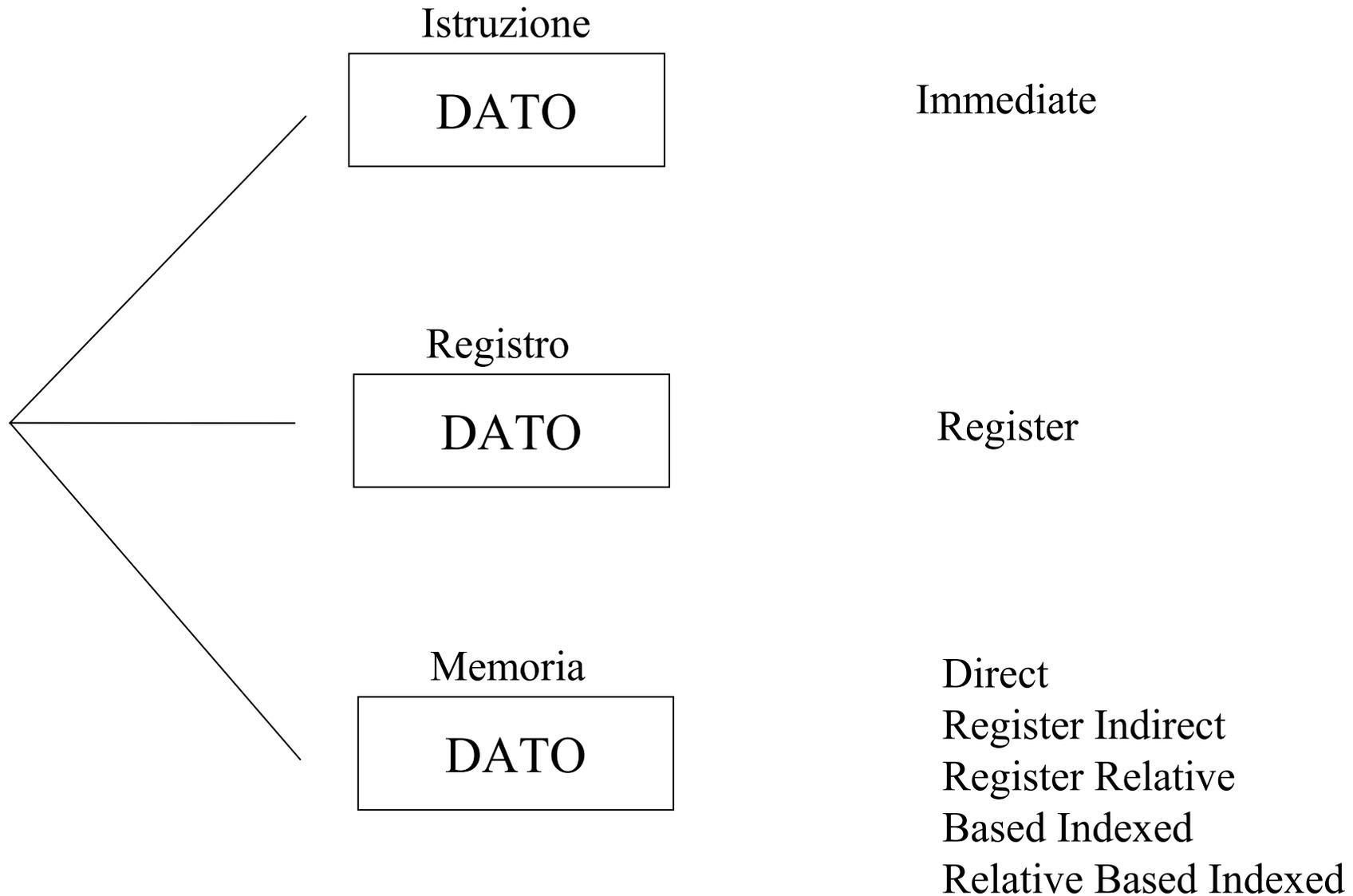
Registri 80386/80486 (visibili al programmatore di applicazioni)



Istruzioni e modalità di indirizzamento

- Il set di istruzioni dell'Assembler Intel 80x86 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, ...

Modi di indirizzamento ai dati nell'8086



Modi di indirizzamento ai dati nell'8086

1. **Immediate**: il dato è lungo 8 o 16 bit e fa parte dell'istruzione
Es. add ax,1000
2. **Direct**: l'indirizzo (effective address, EA) del dato fa parte dell'istruzione.
Questo indirizzo è a 16 bit
Es. add ax, [1000]
3. **Register**: il dato è nel registro specificato dall'istruzione.
Operando a 16 bit: AX, BX, CX, DX, SI, DI, SP, BP
Operando a 8 bit: AL, AH, BL, BH, CL, CH, DL, DH
Es. add ax, bx
4. **Register indirect**: l'indirizzo (EA) del dato è nel registro base BX o in un registro indice (DI o SI)

$$EA = \left\{ \begin{array}{l} (BX) \\ (DI) \\ (SI) \end{array} \right\} \quad \text{Es. add ax, [bx]}$$

Modi di indirizzamento ai dati nell'8086 [2]

5. **Register relative:** l'indirizzo (EA) del dato è dato dalla somma di uno scostamento (a 8 o 16 bit) con il contenuto di un registro base (BX o BP) o di un registro indice (DI o SI)

$$EA = \left\{ \begin{array}{l} (BX) \\ (BP) \\ (DI) \\ (SI) \end{array} \right\} + \text{scostamento}$$

Es. add ax, 1000[bx]

6. **Based indexed:** l'indirizzo (EA) del dato è dato dalla somma del contenuto di un registro base (BX o BP) con il contenuto di un registro indice (DI o SI)

$$EA = \left\{ \begin{array}{l} (BX) \\ (BP) \end{array} \right\} + \left\{ \begin{array}{l} (SI) \\ (DI) \end{array} \right\}$$

Es. add ax, [bx][di]

Modi di indirizzamento ai dati nell'8086 [3]

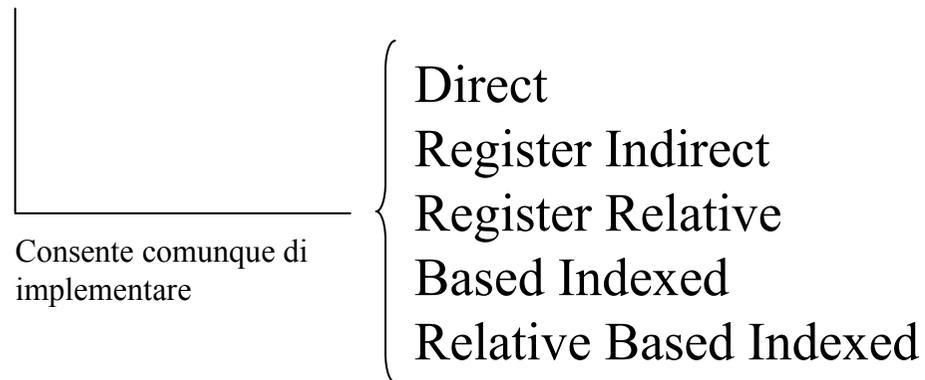
7. **Relative Based Indexed**: l'indirizzo (EA) del dato è dato dalla somma di uno scostamento (a 8 o 16 bit) con il contenuto di un registro base (BX o BP) e di un registro indice (DI o SI)

$$EA = \left\{ \begin{array}{c} (BX) \\ (BP) \end{array} \right\} + \left\{ \begin{array}{c} (SI) \\ (DI) \end{array} \right\} + \text{scostamento}$$

Es. `add ax, 1000[bx][si]`

SI NOTI CHE, NEL CASO DEL MIPS, PER I DATI ABBIAMO SOLO TRE MODALITA' DI INDIRIZZAMENTO:

- IMMEDIATO
- REGISTRO
- BASE e SPIAZZAMENTO



Effective address e Indirizzo fisico

- Nel caso di operando in memoria, quanto specificato è un effective address (EA)
- Noi abbiamo visto che

$$\text{Ind. Fisico} = \text{Ind. Inizio segmento} + \text{EA}$$

↑
Reg. Segmento * 16

↑
Modalità di indirizzamento

Istruzione mov

- Consideriamo l'istruzione mov (move):

mov destination, source

- Questa istruzione copia un dato dall'operando *source* all'operando *destination*
- Posso usare i registri come operandi, l'importante è che siano della stessa ampiezza
- Esempi:

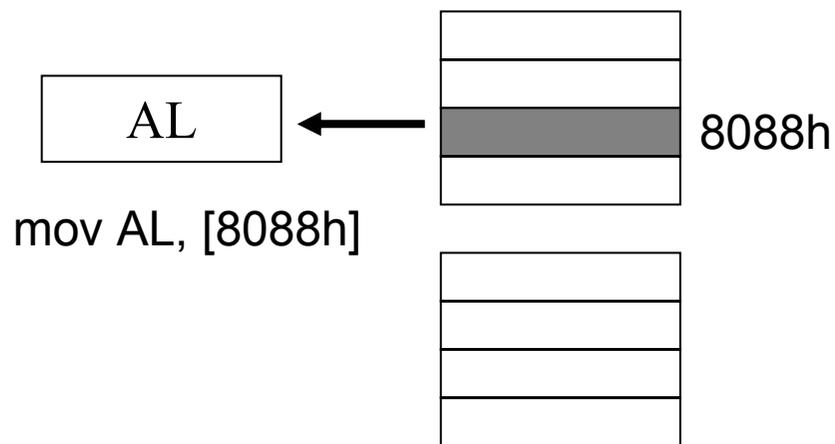
```
mov    ax, bx    ; Copia il valore da BX in AX
mov    dl, al    ; Copia il valore da AL in DL
mov    si, dx    ; Copia il valore da DX in SI
mov    sp, bp    ; Copia il valore da BP in SP
mov    dh, cl    ; Copia il valore da CL in DH
mov    ax, ax    ; Non fa nulla, ma è legale
mov    ax, cs    ; Copia il valore di CS in AX
mov    ds, ax    ; Copia il valore di AX in DS
```

- Es. illegali:

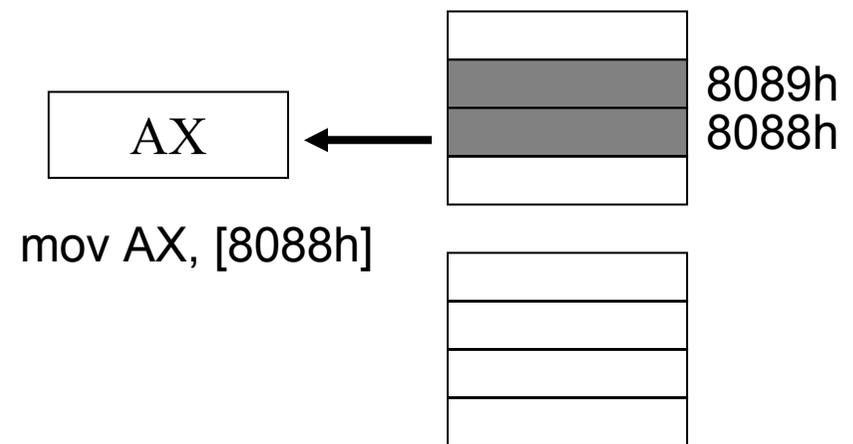
```
mov    ax, bl
mov    al, bx
```

- Attenzione: quando si usano i registri di segmento non si può usare cs come operando destinazione e soltanto uno degli operandi può essere un registro segmento

Accesso al byte e alla parola



Accesso al byte



Accesso alla parola

- Dall' 80386 in poi anche accesso a doppie parole in modo simile

Modi di indirizzamento nell'80386

- Intel ha aggiunto nuovi modi di indirizzamento con l'uscita dell' 80386
- In particolare i modi dell'8086 sono stati generalizzati: mentre prima si potevano usare i registri **bx** e **bp** come registri base ed **si** e **di** come registri indice, l'80386 permette di usare **qualsiasi registro general purpose** a 32 bit come base o indice
- E' poi introdotto il modo *scaled indexed* per semplificare l'accesso agli elementi di array (Possono essere usati solo i nomi dei registri a 32 bit e non di quelli a 16 bit)
- Combinando due registri a 32 bit in un modo di indirizzamento, il primo registro rappresenta il *base register* e il secondo l'*index register*
- E' possibile usare anche lo *stesso registro* sia come base che come indice

Esempi

Se il primo registro (base register) è `ebp` o `esp` l'indirizzo è relativo allo stack segment altrimenti l'indirizzo è relativo al data segment

```
mov    al, [eax][ebx]
mov    al, [ebx+ebx]
mov    al, [edx][ebp]    ; usa DS come default
mov    al, [edi][esi]

mov    al, [ebp+ebx]    ; usa SS come default
mov    al, [esp][ecx]  ; usa SS come default
```

Si può aggiungere un displacement come nell'8086 per ottenere il modo relative based indexed

Scaled Indexed Addressing Modes

- La sintassi di questi modi di indirizzamento è la seguente:

`disp[index*n]`

`[base][index*n]`

`disp[base][index*n]`

Dove “base” e “index” sono registri general purpose a 32 bit e “n” è il valore 1, 2, 4 o 8

- Supponiamo che `ebx` contenga `1000h` e `esi` contenga `4` si ha:

`mov al, 8[ebx][esi*4]` ; copia in `al` il dato nella locazione¹ `1018h`

`mov al, 1000h[ebx][ebx*2]` ; copia in `al` il dato nella locazione¹ `4000h`

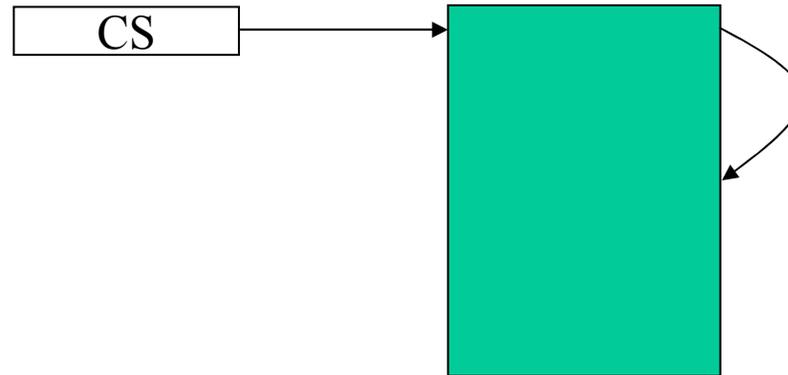
`mov al, 1000h[esi*8]` ; copia in `al` il dato nella locazione¹ `1020h`

¹) è sempre comunque un effective address

Modi di indirizzamento in caso di salto

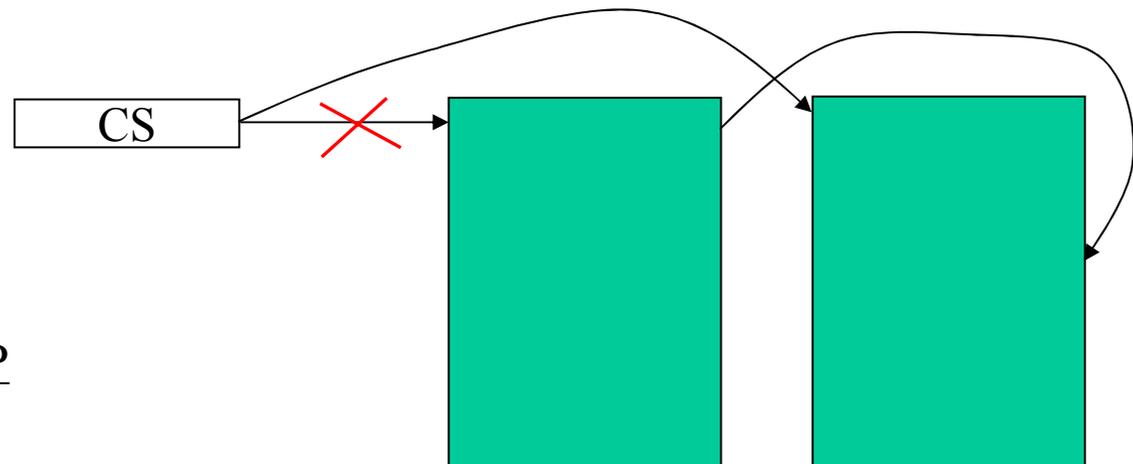
- Dobbiamo specificare un indirizzo fisico a cui saltare

1. Intrasegment



Varia solo IP

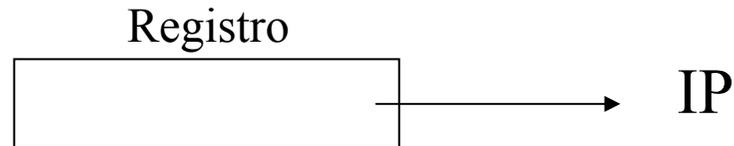
2. Intersegment



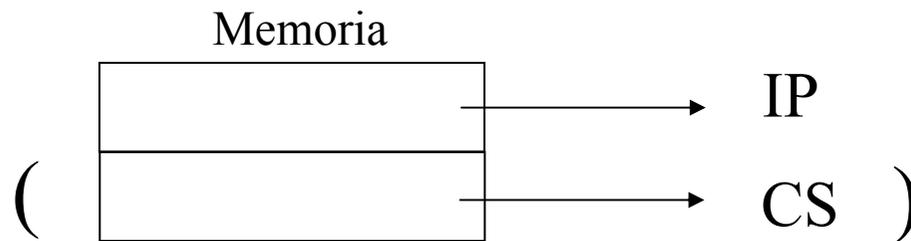
Variano CS e IP

- Come specifico IP [e CS]?

1. Direttamente nell'istruzione (Direct)
2. Con uno dei modi di indirizzamento per i dati (Indirect)



[Register,
solo nell'intrasegment]



[Diretto, Reg. Ind.,
Reg. Rel., Based Index.
Rel. Based Index.]

Modi di indirizzamento in caso di salto

- ***Intrasegment Direct***: l'indirizzo di salto (*effective branch address*) è dato dalla somma di uno scostamento di 8 o 16 bit con il contenuto corrente di IP
- ***Intrasegment Indirect***: l'indirizzo di salto è contenuto in un registro o in una locazione di memoria acceduta con uno dei modi visti, eccetto l'immediato. Il contenuto di IP è sostituito da questo indirizzo.
- ***Intersegment Direct***: viene sostituito il contenuto di IP con una parte dell'istruzione e il contenuto di CS con un'altra parte dell'istruzione (branch da un segmento di codice a un altro segmento)
- ***Intersegment Indirect***: viene sostituito il contenuto di IP e di CS con il contenuto di due parole consecutive in memoria a cui si fa riferimento con i modi di indirizzamento visti, esclusi i modi immediate e register

NB: i salti condizionati possono usare solo Intrasegment con scostamento a 8 bit

Formato delle Istruzioni

Codifica delle istruzioni

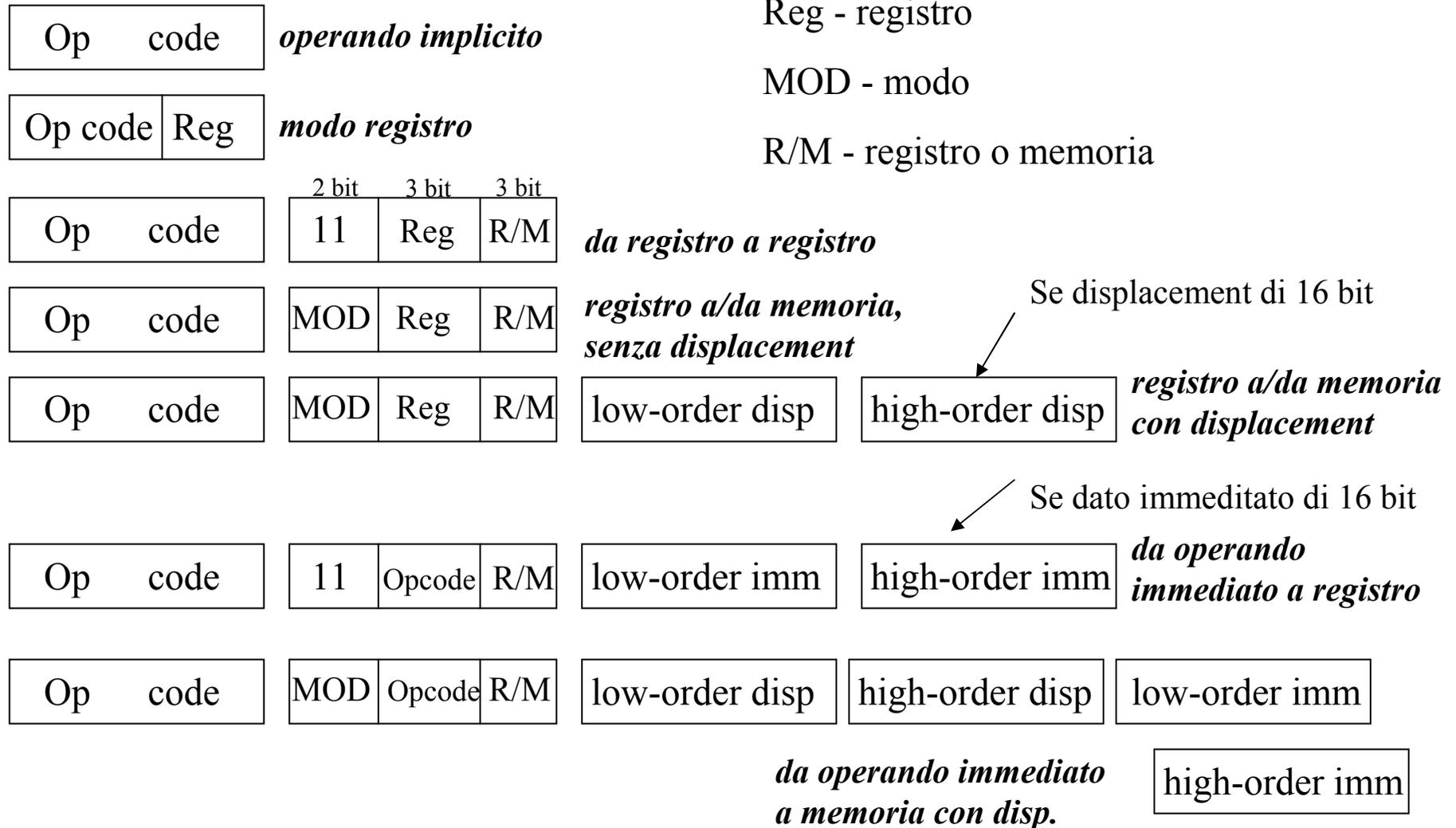
- Codifica delle istruzioni complessa: molti formati diversi
- Nell'8086 le istruzioni possono essere lunghe da 1 a 6 byte
- Nelle architetture successive le istruzioni possono essere lunghe fino a 17 byte
- Un'istruzione è sempre formata da un primo (a volte unico) byte che contiene il *codice operativo*
- Il codice operativo nel primo byte di alcune istruzioni include il modo di indirizzamento
- Altre istruzioni utilizzano *un ulteriore byte per il codice operativo*, spesso chiamato "*mod-reg-r/m*", con informazioni relative al modo di indirizzamento (in particolare per istruzioni che accedono in memoria)
- Gli altri byte aggiuntivi dell'istruzione possono contenere dati immediati o indirizzi

Formato delle istruzioni nell'8086

- Al byte o ai byte che contengono codice operativo e modo di indirizzamento possono seguire :
 - Nessun byte addizionale
 - 2 byte di un indirizzo effettivo (EA) [solo nel caso di indirizzamento diretto]
 - 1 o 2 byte di un displacement
 - 1 o 2 byte di un operando immediato
 - 1 o 2 byte di un displacement seguito da 1 o 2 byte di un operando immediato
 - 2 byte di un displacement seguito da 2 byte di un indirizzo di segmento [cfr. salto intersegment diretto]

NB: se un immediato, EA o displacement è lungo 2 bytes, il byte meno significativo appare per primo in memoria (indirizzo minore)

Esempi di formati delle istruzioni nell'8086



Bit speciali di op code e campo Reg nell'8086

- Nel codice operativo ci sono dei bit speciali, in particolare:
 - Bit W: indica se l'istruzione opera su un byte ($W=0$) oppure su una parola ($W=1$)
 - Bit D: indica se il registro indicato nel campo Reg (nelle istruzioni in cui un operando è un registro) è da considerarsi come operando sorgente ($D=0$) o come operando destinazione ($D=1$)

Campo Reg: 
codifica dei registri

Reg	W = 1	W=0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

- Bit S: Nelle istruzioni add, subtract, compare immediate to register/memory

Indica se un immediato a 8 bit con segno viene esteso a 16 bit
(consentendo di risparmiare un byte nel caso di numeri piccoli):

S : W	
0 0	operazione a 8 bit
0 1	operazione a 16 bit con op. immediato a 16 bit
1 1	operazione a 16 bit con operando a 8 bit – segno esteso

Modi di indirizzamento con i campi MOD e R/M

- Indichiamo nella tabella i modi di indirizzamento per le diverse combinazioni dei campi MOD e R/M
- D8 sta per “displacement a 8 bit” (il cui segno è esteso a 16 bit, in esecuzione) e D16 sta per “displacement a 16 bit”

R/M	MOD	00	01	10	11
					w=0 w=1
000 segment addr.		[BX]+[SI] DS	[BX]+[SI]+D8 DS	[BX]+[SI]+D16 DS	AL AX
001 segment addr.		[BX]+[DI] DS	[BX]+[DI]+D8 DS	[BX]+[DI]+D16 DS	CL CX
010 segment addr.		[BP]+[SI] SS	[BP]+[SI]+D8 SS	[BP]+[SI]+D16 SS	DL DX
011 segment addr.		[BP]+[DI] SS	[BP]+[DI]+D8 SS	[BP]+[DI]+D16 SS	BL BX
100 segment addr.		[SI] DS	[SI]+D8 DS	[SI]+D16 DS	AH SP
101 segment addr.		[DI] DS	[DI]+D8 DS	[DI]+D16 DS	CH BP
110 segment addr.		D16 DS	[BP]+D8 SS	[BP]+D16 SS	DH SI
111 segment addr.		[BX] DS	[BX]+D8 DS	[BX]+D16 DS	BH DI

NB: per default DS è reg. segm., a meno che non si usi BP come base [SS è reg. segm.]

Campi MOD e R/M

- I campi MOD (2 bit) e R/M (3 bit) combinati insieme danno il modo di indirizzamento (vedi lucido seguente!)

MOD	Significato
00	Indirizzamento alla memoria senza displacement, tranne nel caso in cui R/M vale 110 con il quale si intende un indirizzamento diretto
01	Indirizzamento alla memoria con displacement da 8 bit (1 byte che segue il mod/reg/rm byte)
10	Indirizzamento alla memoria con displacement da 16 bit (2 byte che seguono il mod/reg/rm byte)
11	Il campo R/M denota un registro con la stessa codifica vista per il campo Reg

NB: un operando immediato è “implicato” dal codice operativo

Segment override prefix

- Nella tabella precedente comparivano come registri segmento soltanto DS e SS
- Per utilizzare gli altri segmenti, esiste una istruzione ad un byte che indica il segmento da utilizzare nella istruzione seguente (al posto di quello di default)

001 REG 110

2 bit: 00: ES
01: CS
10: SS
11: DS

- Esistono comunque delle eccezioni, p.es. CS è sempre il registro segmento nelle istruzioni di salto, SS quando si usa Stack Pointer [istruzioni PUSH e POP]

Esempio: istruzione ADD (1)

- L'istruzione add può avere diverse forme:

```
add    reg, reg
add    reg, memory
add    memory, reg
add    reg, constant
add    memory, constant
```

- Tutte aggiungono il secondo operando al primo lasciando il risultato nel primo operando

- Esempio:

add cl, bh

Istruzione di 2 byte



formato 0 0 0 0 0 0 DW MOD REG R/M

in binario 0 0 0 0 0 0 1 0 11 001 111

REG è operando
destinazione

CL BH

Il campo R/M
denota un registro

Operandi di 8 bit

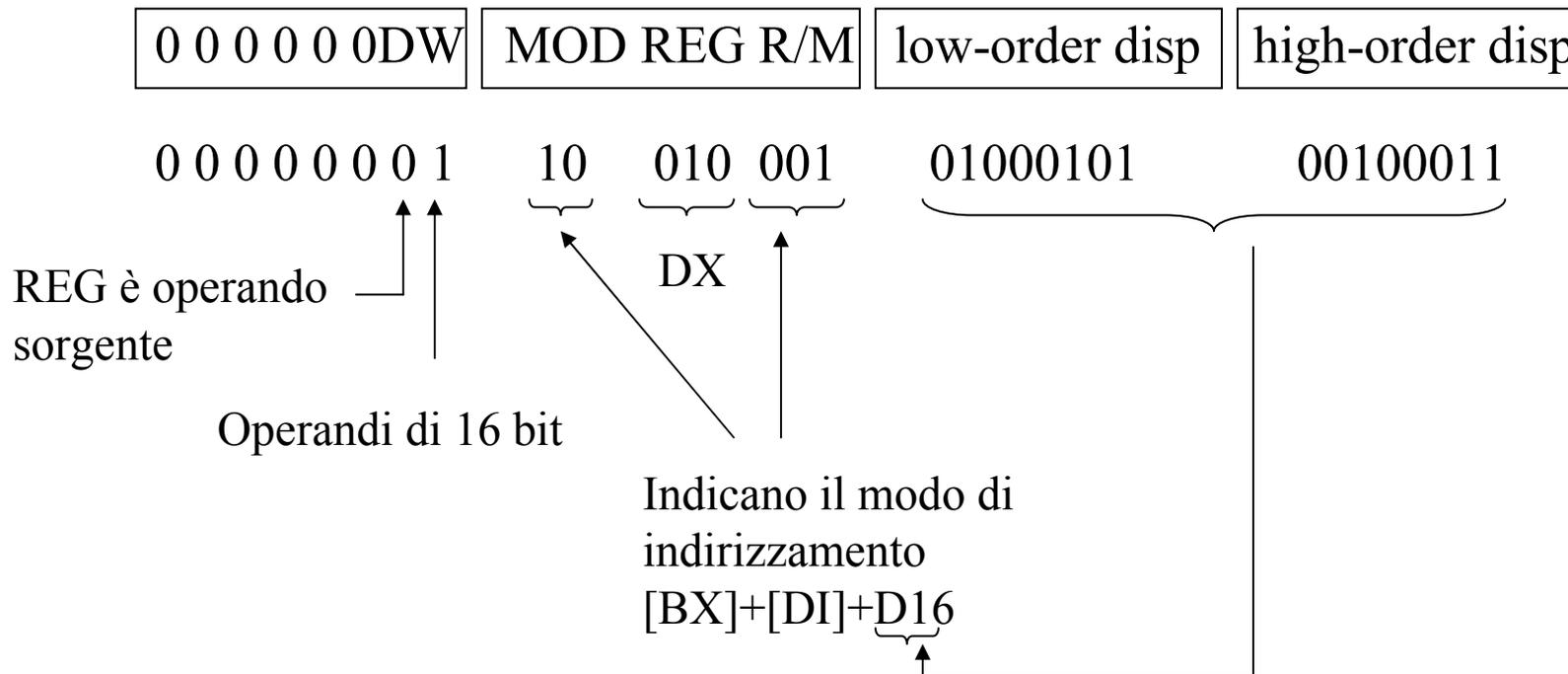
Nota: esiste un altro
formato equivalente.
Quale? (suggerimento:
si cambi la direzione D)

Esempio: istruzione ADD (2)

Vediamo ora l'istruzione

add mem, dx

Istruzione di 4 byte



Istruzioni aritmetiche, logiche e trasferimento dati

- Operano su 1 o al massimo 2 operandi
- Le combinazioni di operandi nelle istruzioni aritmetiche, logiche e trasferimento dati nell'80x86 sono le seguenti

Tipo dell' operando sorgente / destinazione	Tipo del secondo operando sorgente
Registro	Registro
Registro	Immediato
Registro	Memoria
Memoria	Registro
Memoria	Immediato

- Nota che non esiste la modalità memoria-memoria
- Gli operandi immediati possono avere lunghezza 8 o 16 bit fino al 80286 e anche 32 bit dall'80386 in poi

Un confronto fra Assembler MIPS e Assembler Intel 80x86

- L'80x86 ha un numero limitato di registri general purpose (8) rispetto al MIPS
- La codifica delle istruzioni nell'80x86 è complessa: esistono molti formati diversi con dimensioni variabili
- Nel MIPS esistono pochi formati: formato-R, formato-I e formato-J, con dimensione fissa di 32 bit
- Le istruzioni aritmetiche e logiche nell'80x86 hanno sempre un operando che fa sia da sorgente che da destinazione, mentre il MIPS consente di avere registri distinti per sorgenti e destinazione (ciò è anche legato al numero di registri disponibili)

Un primo confronto fra Assembler MIPS e Assembler Intel 80x86 (continua)

- Nell'80x86 uno degli operandi può essere in memoria, a differenza del MIPS in cui solo load e store fanno riferimento alla memoria
- Nell'80x86 i modi di indirizzamento relativi ai dati in memoria sono svariati (ad esempio, nell'8086 sono già 5) contro i 3 visti nel caso del MIPS
- Nel MIPS esistono solo 2 modi di indirizzamento per i salti (PC-relative e pseudodiretto), mentre già l'8086 offre 4 modi
- Da questa analisi sono emerse alcune delle differenze fra un'architettura di tipo **RISC** (*Reduced Instruction Set Computer*) come il MIPS e un'architettura di tipo **CISC** (*Complex Instruction Set Computer*) come l'Intel