Linguaggio Assembler Intel 80x86

Calcolatori Elettronici B

a.a. 2004/2005

Massimiliano Giacomin

Scopi principali

- Studiare un ulteriore esempio di linguaggio assembler
- Saper svolgere semplici programmi
- Apprezzare le principali differenze CISC/RISC
  (e in particolare, apprezzare le differenze MIPS/8086)
- Famiglia Intel: la più diffusa!

Un po' di storia

- **1978:** architettura 8086, a 16 bit, con registri a 16 bit. 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, vengono aggiunte per questo nuove istruzioni (gestione del modo protetto).
- **1985:** architettura 80386, a 32 bit (registri e spazio di indirizzamento a 32 bit), nuovi modi di indirizzamento e nuove istruzioni. Supporto alla paginazione oltre che alla segmentazione degli indirizzi.
- **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:** Espansione delle architetture Pentium con set di istruzioni MMX per accelerare le applicazioni legate alla multimedialità e alle comunicazioni
- **Oggi:** Pentium III (e Pentium IV): miglioramento tecnologia per aumentare prestazioni. Nuovo set di istruzioni Internet SSE per accelerare applicazioni legate a Internet.

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 la differenza nelle architetture successive rispetto all'8086

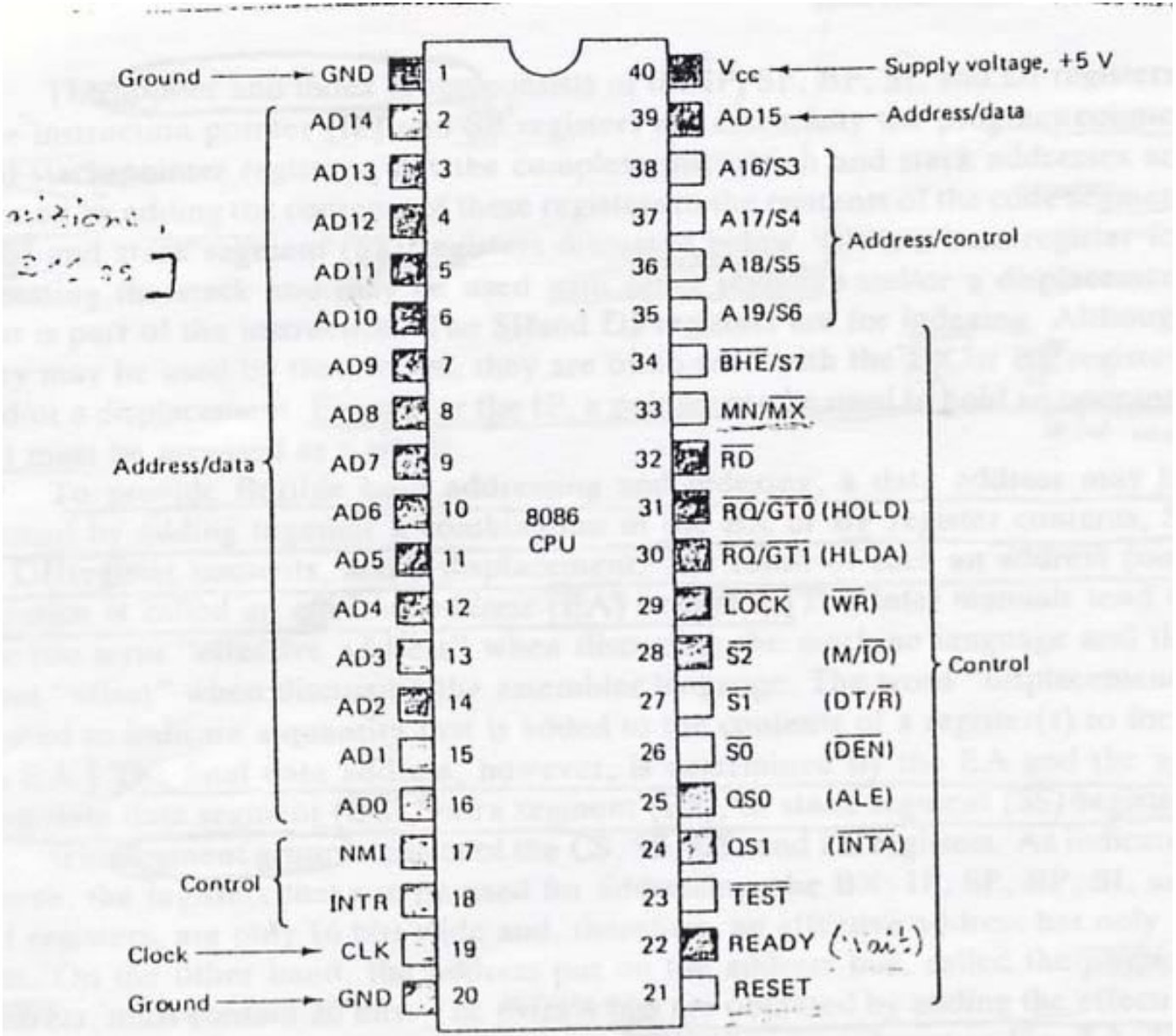
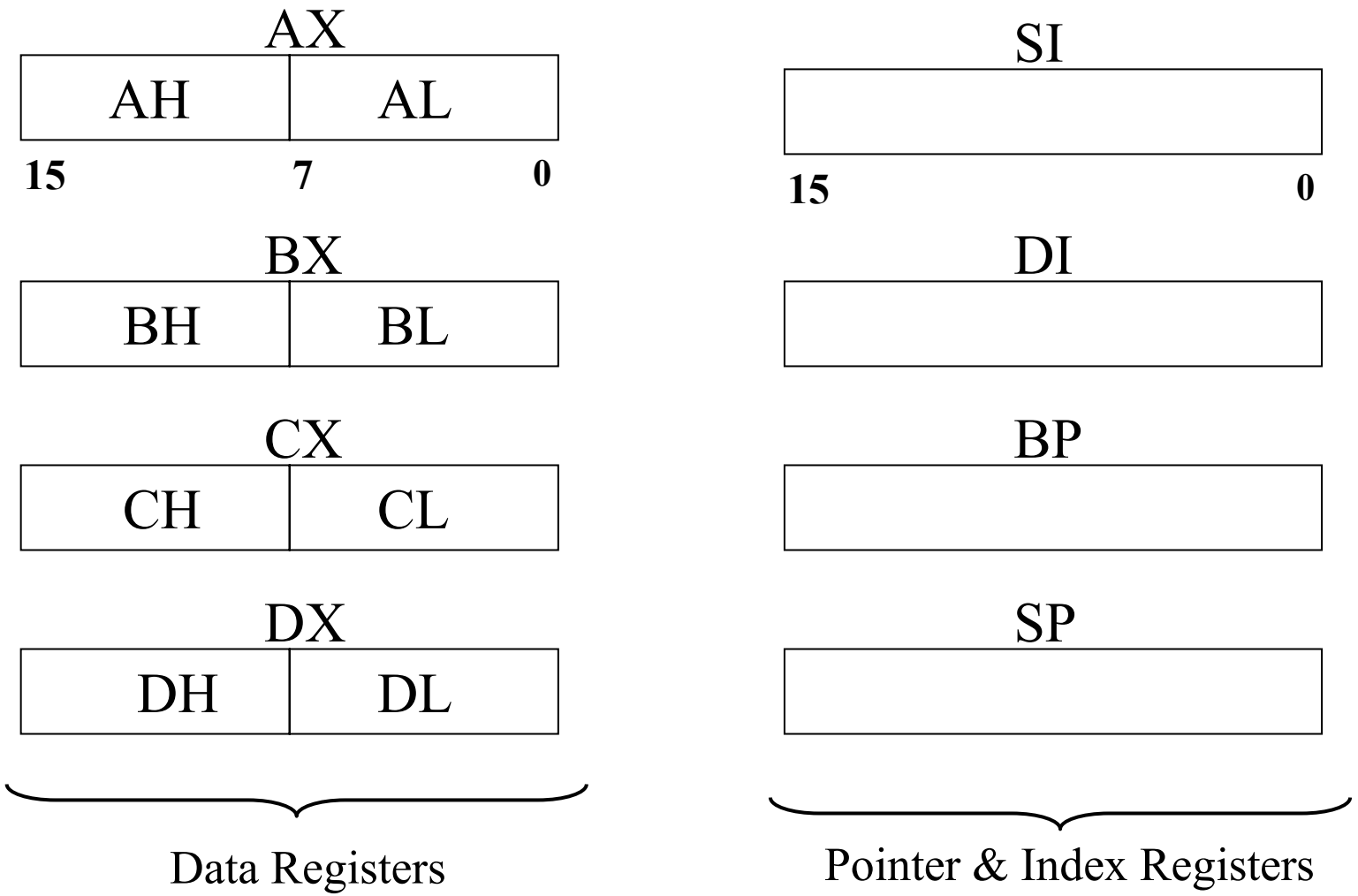


Figure 2-1 8086 pin assignments. (Reprinted by permission of Intel Corporation. Copyright 1981.)

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



I registri AX, BX, CX, DX

- I registri AX, BX, CX e DX sono registri che servono per memorizzare operandi e risultati di operazioni, fungono da registri aritmetici
- Il byte meno significativo (L) e il byte più significativo (H) possono essere acceduti separatamente: cioè si possono usare AH e AL, BH e BL, etc.
- BX, CX e DX hanno anche altri ruoli:
  - BX può essere usato come registro base negli indirizzi
  - CX viene usato come contatore implicito in certe istruzioni
  - DX viene usato per indirizzi di I/O in certe operazioni di I/O

I registri puntatore e indice

- I registri SP, BP, SI, DI servono per 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).

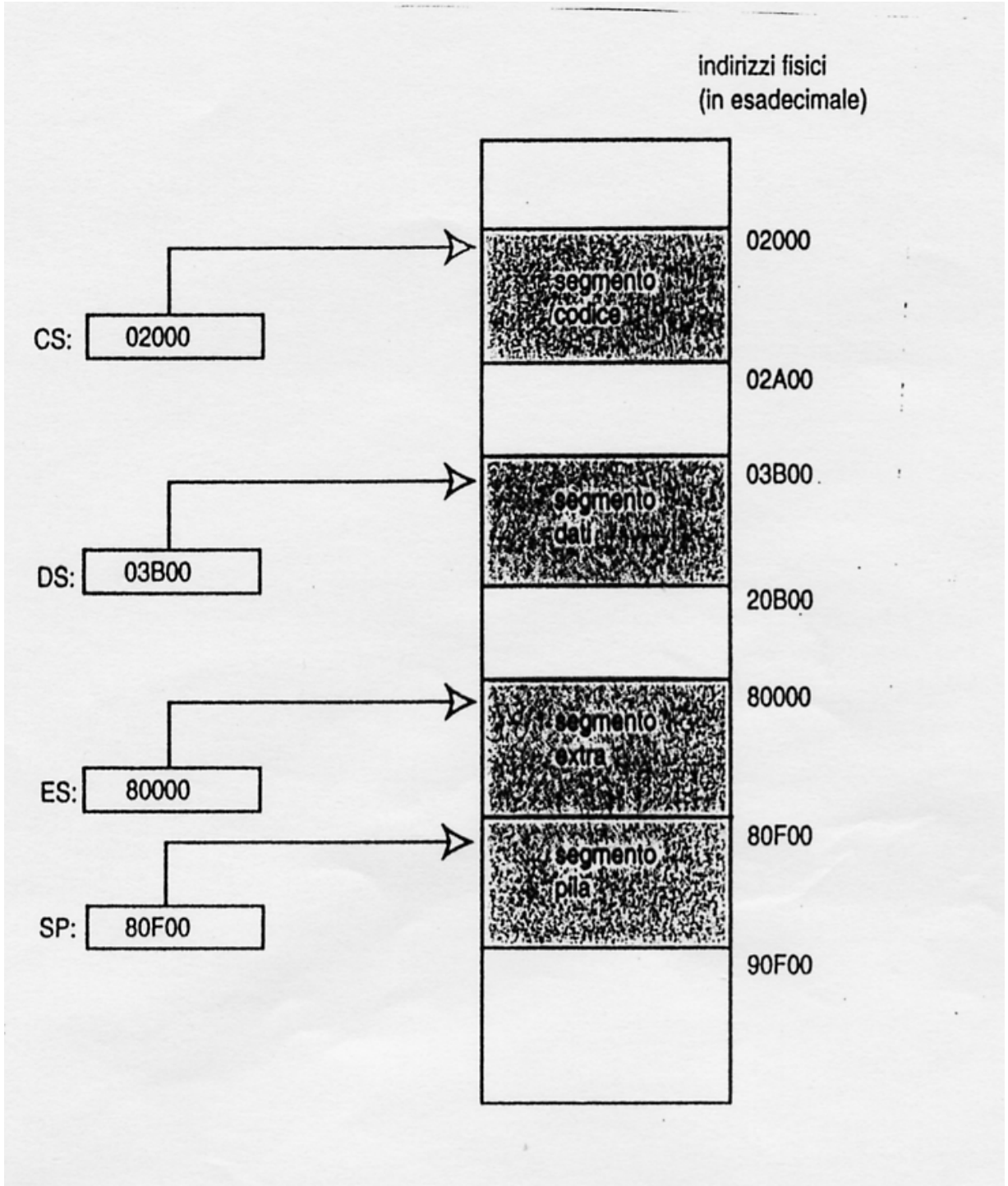
P.es. (BP)+(SI)

BASE INDICE

Tutti i registri possono anche essere usati per memorizzare operandi, ma non è possibile accedere al byte.

I registri segmento

- 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)
- I registri segmento sono usati in due modalità diverse chiamate *real mode* e *protected mode*. Vedremo solo la modalità real mode (vedi dopo "calcolo dell'indirizzo fisico")


indirizzi fisici
(in esadecimale)
CS:
02000
DS:
03B00
ES:
80000
SP:
80F00
segmento codice
segmento dati
segmento extra
segmento pila
02000
02A00
03B00
20B00
80000
80F00
90F00

Memoria

- 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)

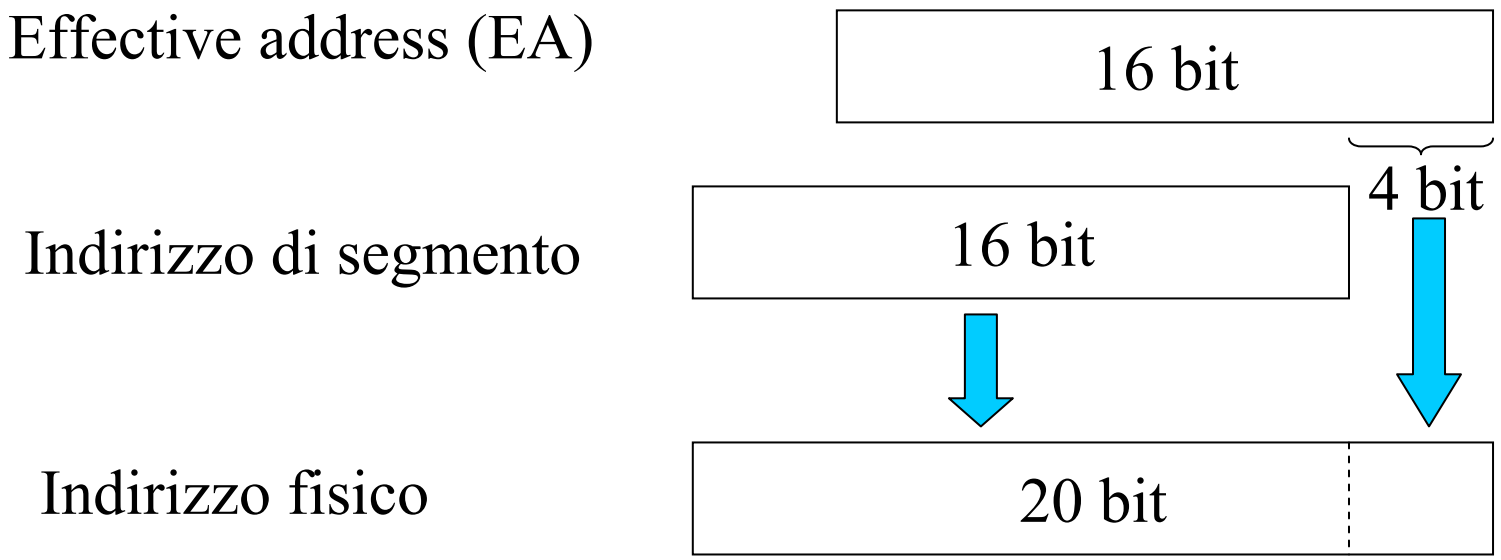
  [IP] = 341B (esadecimale)

  la prossima istruzione sarà prelevata dalla posizione

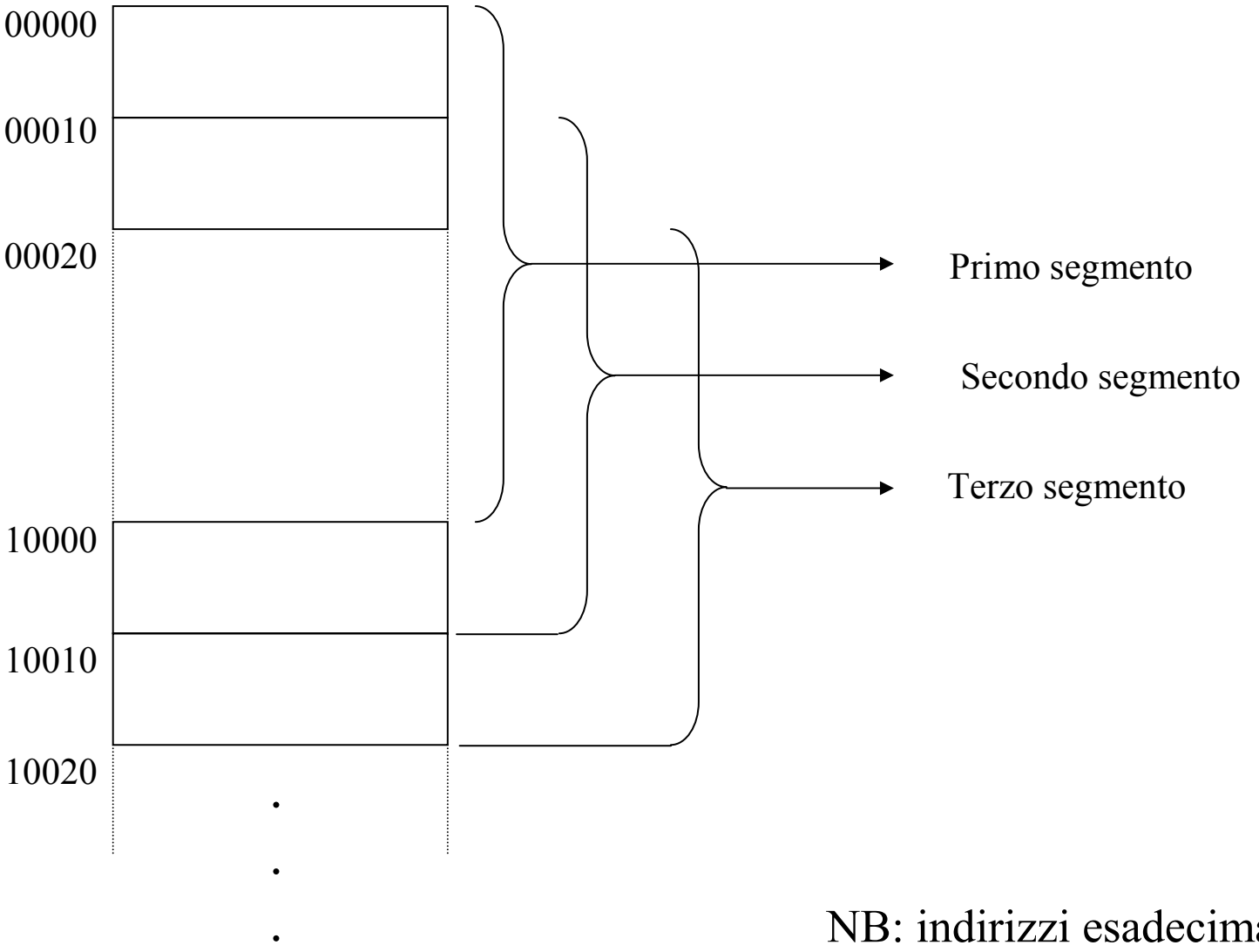
  341B + offset o effective address

  123A0 = indirizzo di inizio segmento

  157BB indirizzo fisico


Effective address (EA)
16 bit
4 bit
Indirizzo di segmento
16 bit
Indirizzo fisico
20 bit

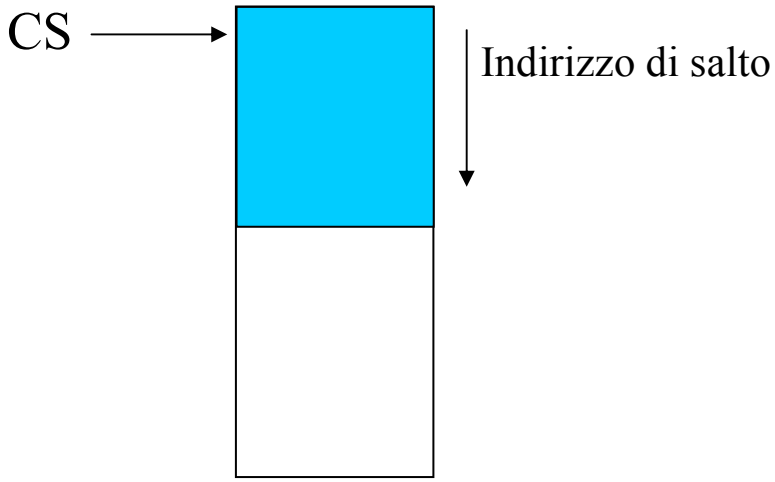
Segmentazione della memoria

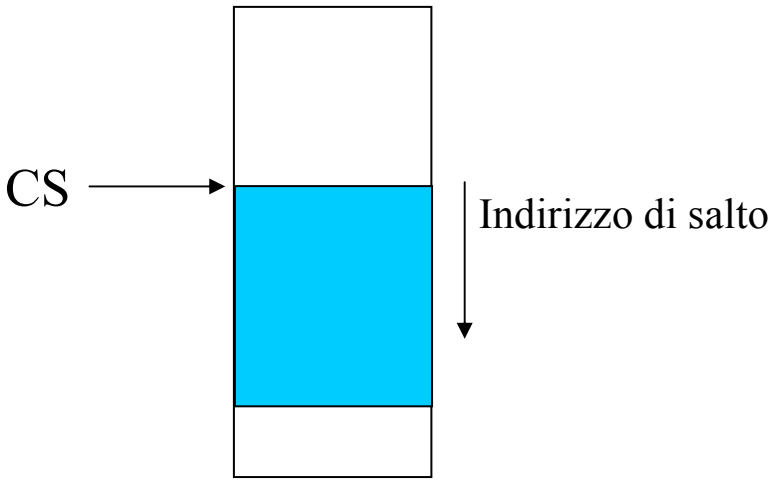


NB: indirizzi esadecimali

Segmentazione della memoria: benefici

1. Permette di indirizzare 1 MB (2^{20} Bytes) con i registri a 16 bit
2. Consente di disporre per un programma di uno spazio per il codice, dati, stack superiore a 64K (utilizzando più segmenti)
3. Facilita l'utilizzo di aree separate codice-dati-stack (gestione della memoria nel caso della multiprogrammazione)
4. Facilita la rilocazione di un programma nel caso della multiprogrammazione





Il registro PSW

- 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

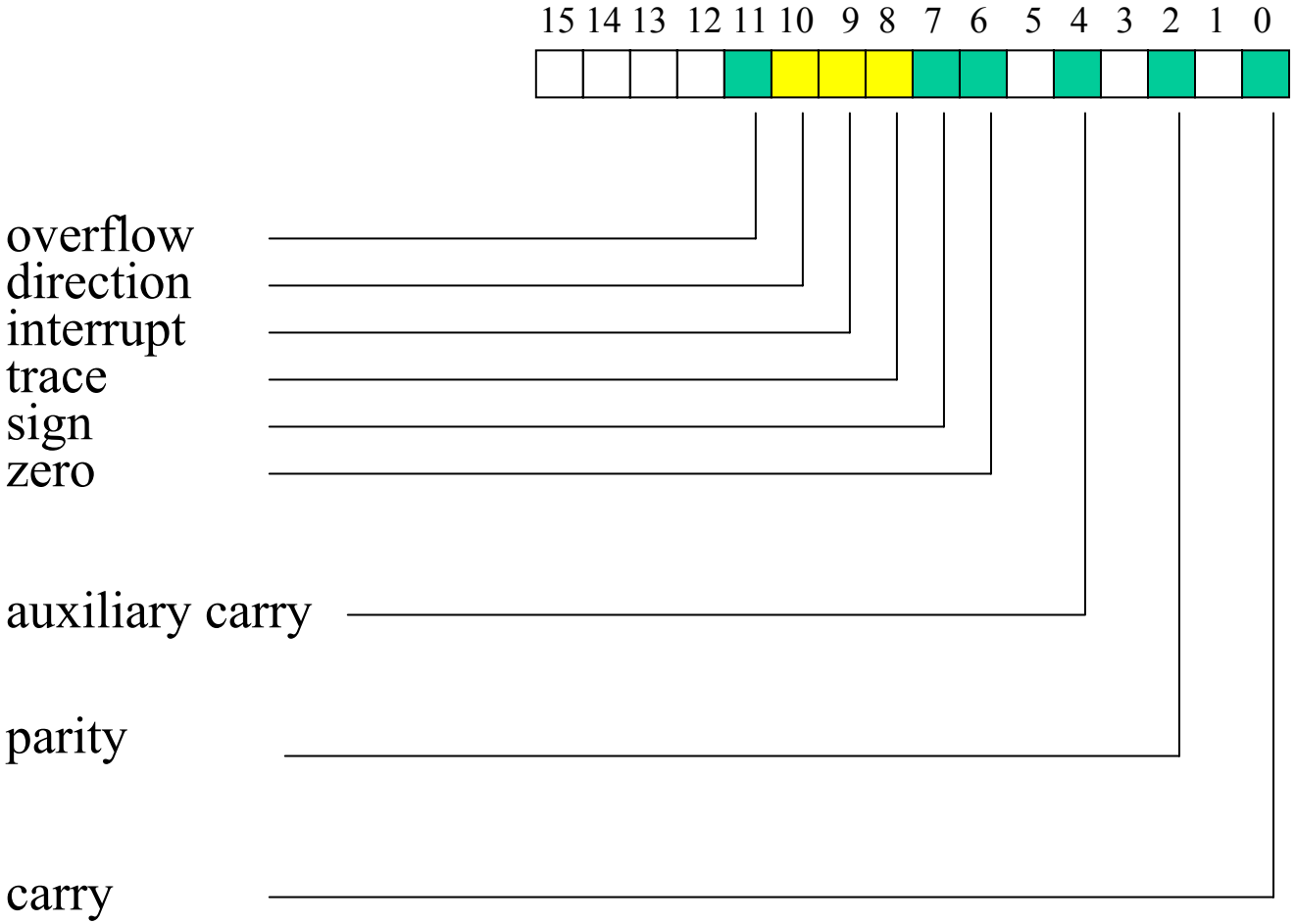
  SF: indicatore di *segno*. E' posto a 1 quando il risultato di una operazione logico-aritmetica è un numero negativo (= MSB del risultato)

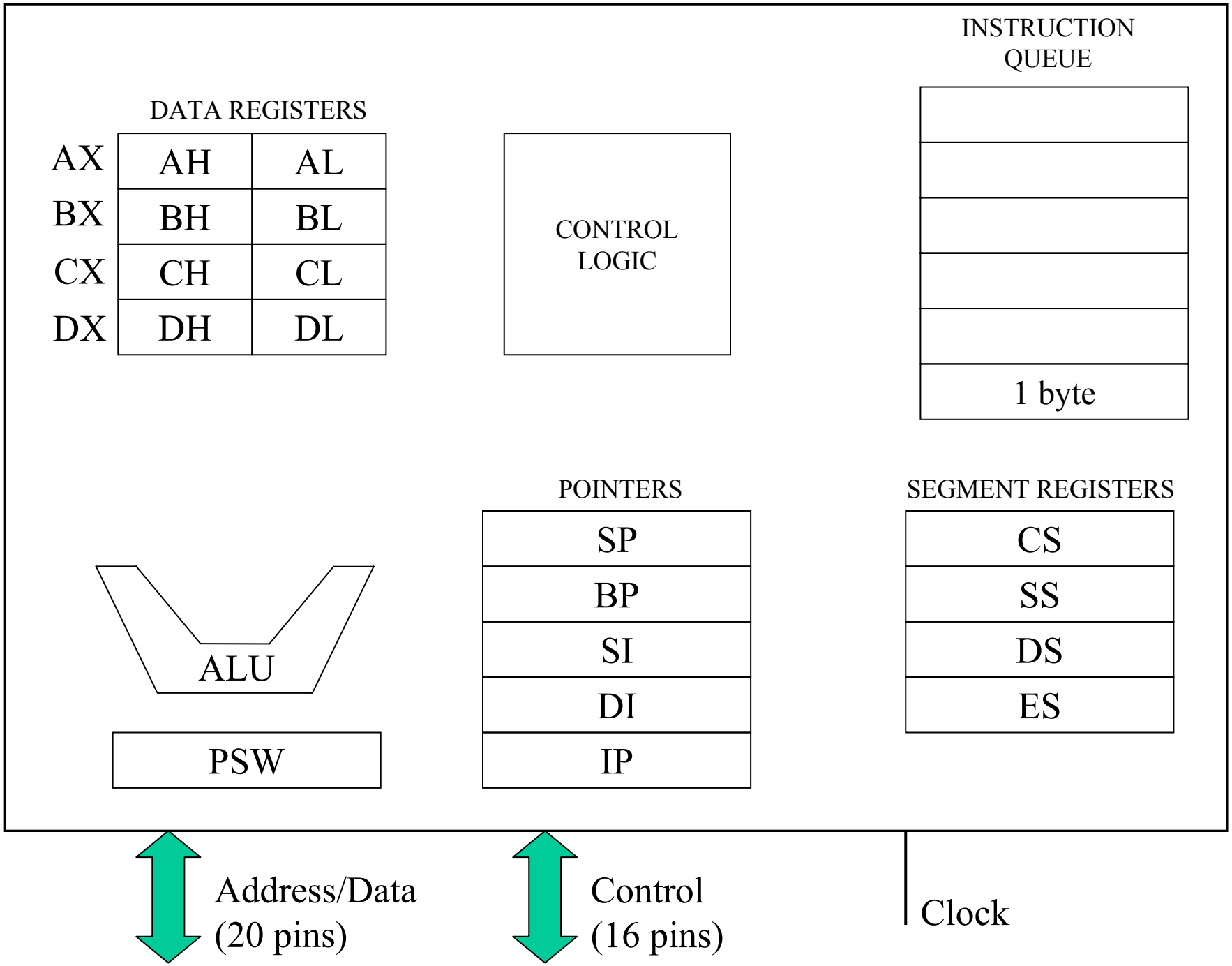
  ZF: indicatore di *zero*. E' posto a 1 quando il risultato di una operazione logico-aritmetica vale 0

  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

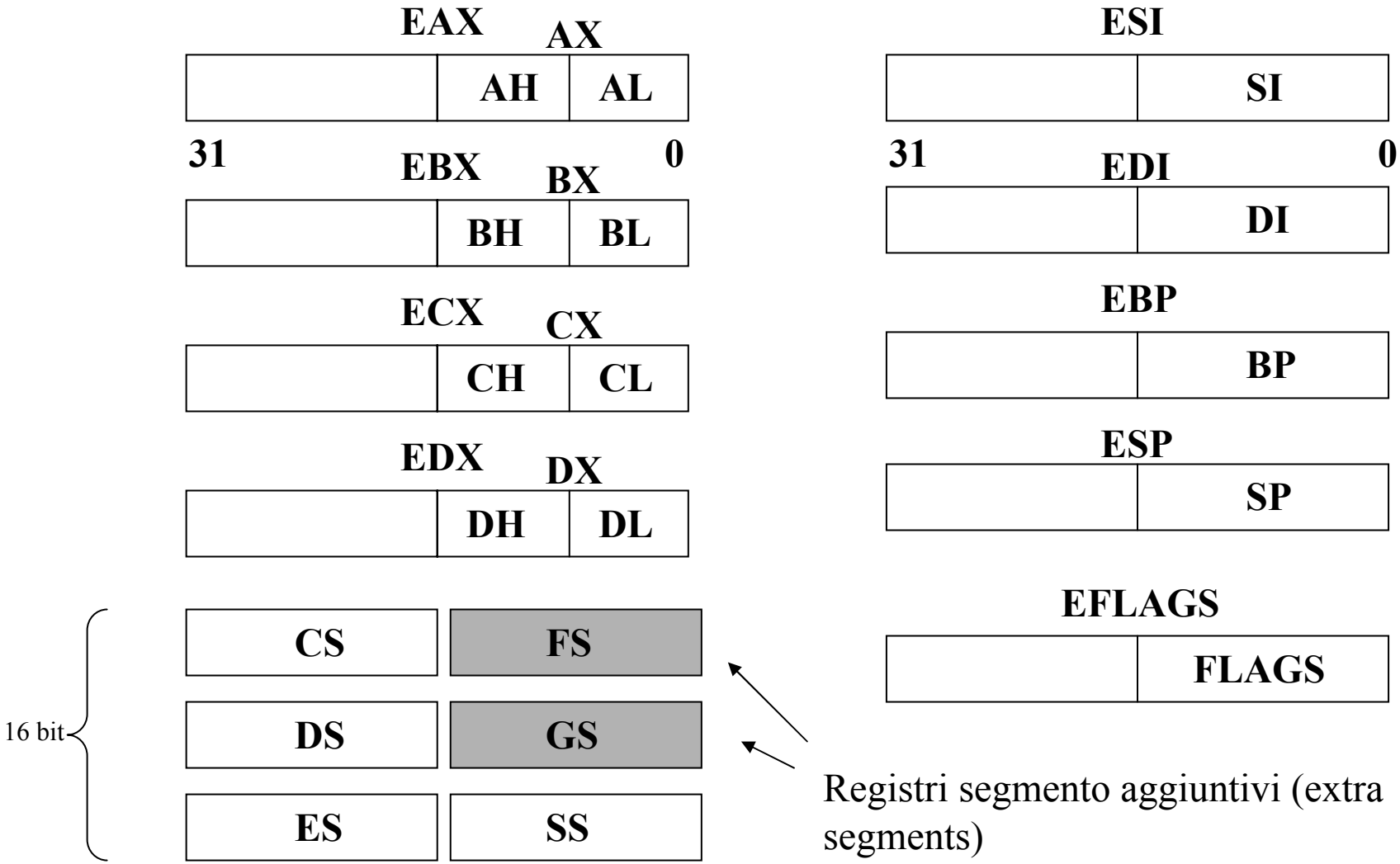



INSTRUCTION QUEUE
DATA REGISTERS
AX AH AL
BX BH BL
CX CH CL
DX DH DL
CONTROL LOGIC
1 byte
POINTERS
SP
BP
SI
DI
IP
SEGMENT REGISTERS
CS
SS
DS
ES
ALU
PSW
Address/Data (20 pins)
Control (16 pins)
Clock

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
- Nel registro eflags altri bit significativi
- 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)



Ampiezza dei BUS

Table 17: 80x86 Processor Data Bus Sizes

Processor	Data Bus Size
8088	8
80188	8
8086	16
80186	16
80286	16
80386sx	16
80386dx	32
80486	32
80586 class/ Pentium (Pro)	64

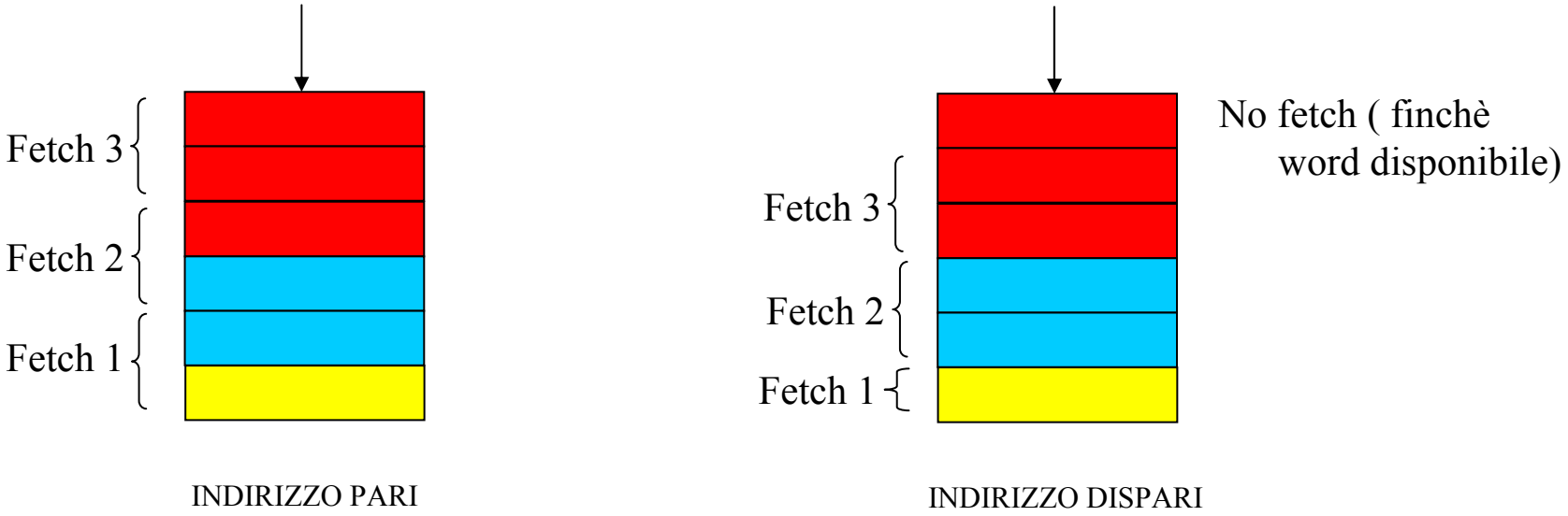
Table 18: 80x86 Family Address Bus Sizes

Processor	Address Bus Size	Max Addressable Memory	In English!
8088	20	1,048,576	One Megabyte
8086	20	1,048,576	One Megabyte
80188	20	1,048,576	One Megabyte
80186	20	1,048,576	One Megabyte
80286	24	16,777,216	Sixteen Megabytes
80386sx	24	16,777,216	Sixteen Megabytes
80386dx	32	4,294,976,296	Four Gigabytes
80486	32	4,294,976,296	Four Gigabytes
80586 / Pentium (Pro)	32	4,294,976,296	Four Gigabytes

Fetch delle istruzioni

- EU: Execution Unit [Comprende ALU e Control Unit]
- BIU: Bus Interface Unit [Gestisce coda di prefetch a 6 bytes]

Fetch dei successivi 2 bytes non appena il BUS è libero
Coda svuotata in caso di salto eseguito
Risparmio di tempo nel caso di istruzione ad indirizzo pari



Istruzioni Linguaggio Macchina

• Formato generico: Codice operativo + Operandi

• 8086: Limitazione a due operandi (sorgente + destinazione)
• Uno dei due operandi può essere in memoria ma non entrambi

Modalità di indirizzamento:

come un operando è specificato

• Indirizzamento ai dati
• Indirizzamento nei salti

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)

$$EA = \begin{Bmatrix} (BX) \\ (DI) \\ (SI) \end{Bmatrix}$$

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)

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

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)

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

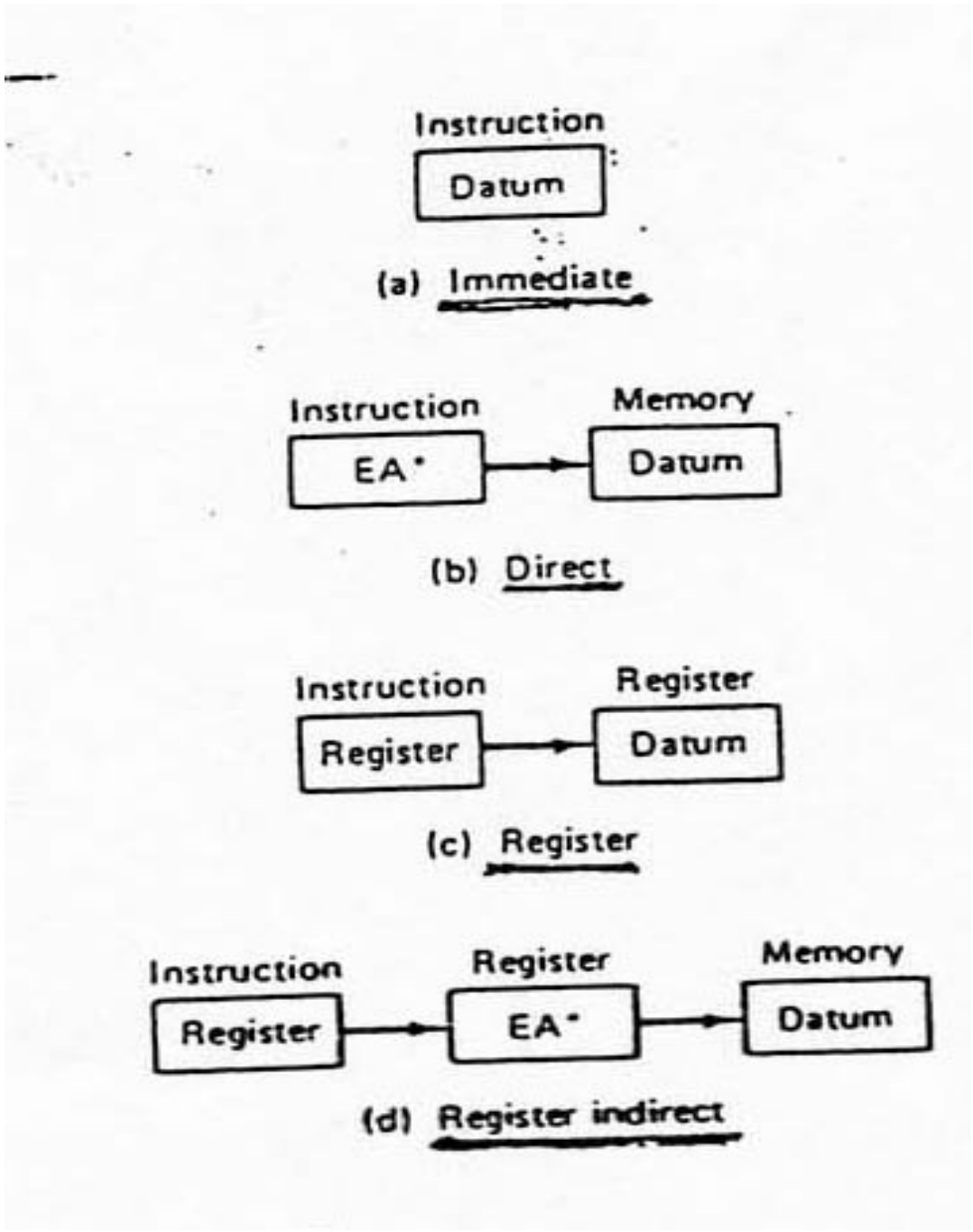
Es. add ax, [bx][di]

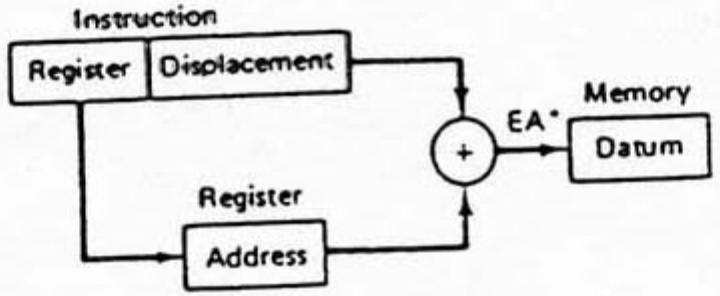
Modi di indirizzamento ai dati nell'8086 [3]

7. **<u>Relative Based Indexed:</u>** 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)

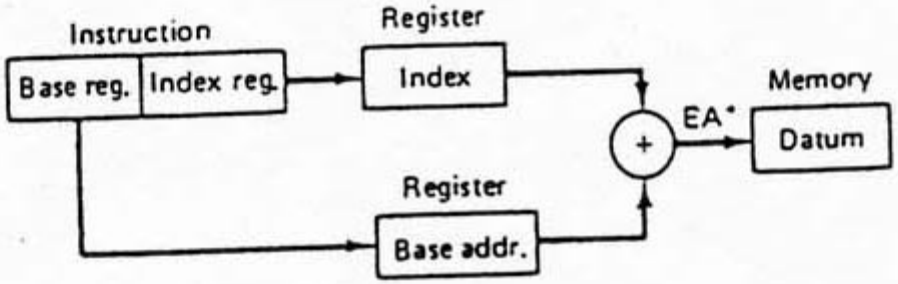
$$EA = \left\{ \begin{matrix} (BX) \\ (BP) \end{matrix} \right\} + \left\{ \begin{matrix} (SI) \\ (DI) \end{matrix} \right\} + \text{scostamento}$$

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

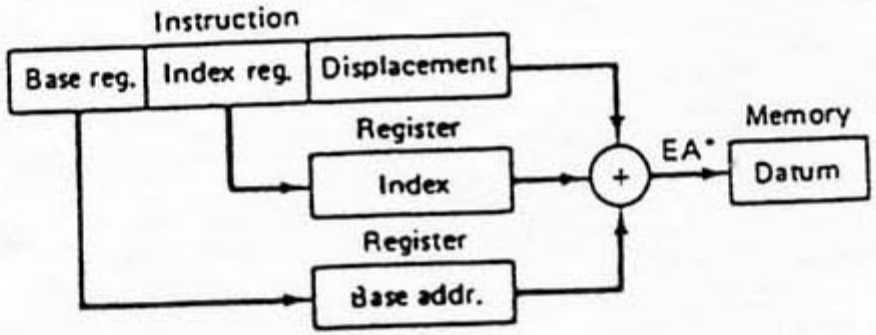

Instruction
Datum
(a) Immediate
Instruction
Memory
EA*
Datum
(b) Direct
Instruction
Register
Register
Datum
(c) Register
Instruction
Register
Register
EA*
Memory
Datum
(d) Register indirect



(e) Register relative

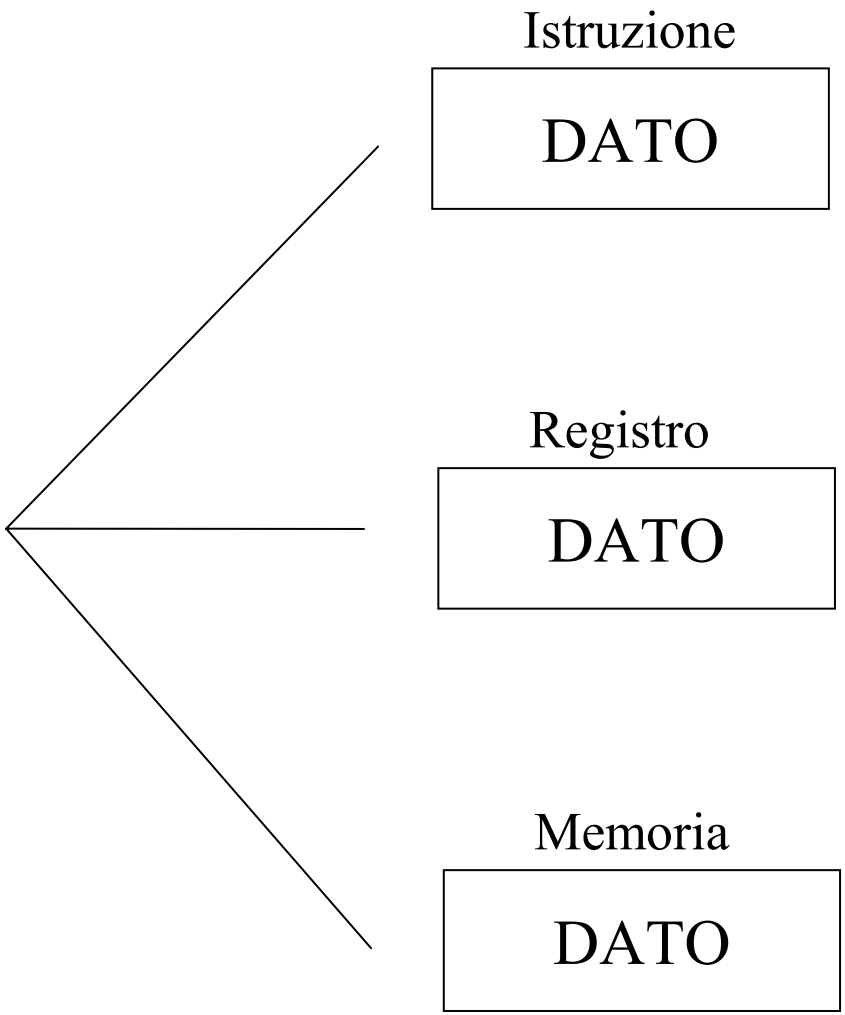


(f) Based indexed



(g) Relative based indexed

*EA is added to 16_{10} times the contents of the appropriate segment register.



Immediate

Register

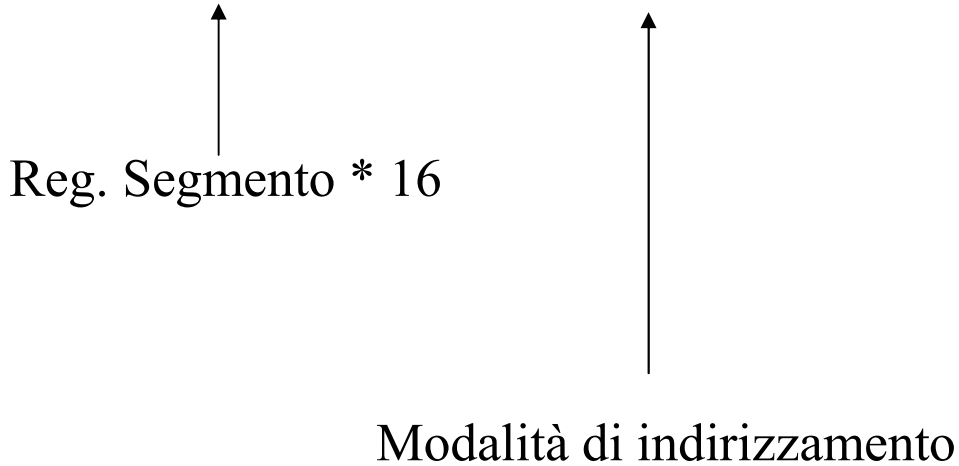
Direct
Register Indirect
Register Relative
Based Indexed
Relative Based Indexed

Effective address e Indirizzo fisico

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

- Noi abbiamo visto che

  Ind. Fisico = Ind. Inizio segmento + EA



Calcolo dell'indirizzo fisico: esempi

Supponiamo: [BX] = 0158 [DI] = 10A5 Disp = 1B57 [DS] = 2100
dove [DS] è usato come indirizzo di segmento
Vediamo quali sono gli effective address (EA) e gli indirizzi fisici nei vari casi:

Immediate: il dato è nell'istruzione – nessun EA

Direct: EA = 1B57
Indirizzo fisico = 21000 + 1B57 = 22B57

Register: il dato è nel registro – nessun EA

Register indirect (con BX):
EA = 0158
Indirizzo fisico = 21000 + 0158 = 21158

Register relative (con BX):
EA = 0158 + 1B57 = 1CAF
Indirizzo fisico = 21000 + 1CAF = 22CAF

Based indexed (con BX e DI)
EA = 0158 + 10A5 = 11FD
Indirizzo fisico = 21000 + 11FD = 221FD

Relative based indexed (con BX e DI)
EA = 0158 + 10A5 + 1B57 = 2D54
Indirizzo fisico = 21000 + 2D54 = 23D54

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:

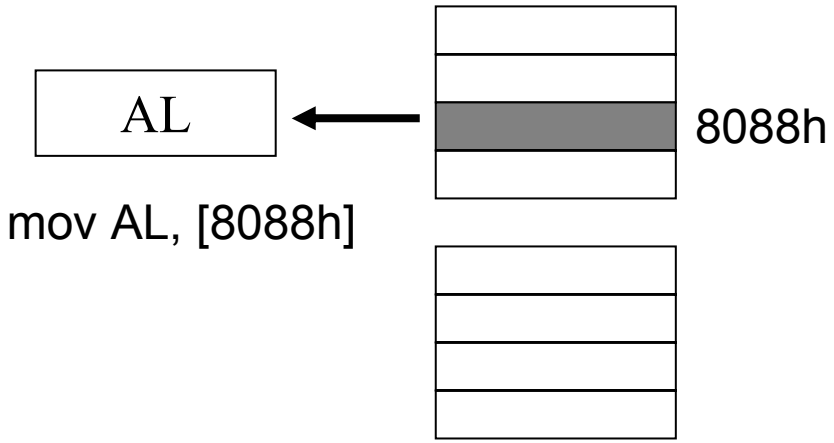
```
mov     ax, bx      ; Copia il valore da BX in AX
mov     dl, al      ; Copia il valore da AL in DL
mov     si, dx      ; Copia il valore da DX in SI
mov     sp, bp      ; Copia il valore da BP in SP
mov     dh, cl      ; Copia il valore da CL in DH
mov     ax, ax      ; Non fa nulla, ma è legale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

- Es. illegali:

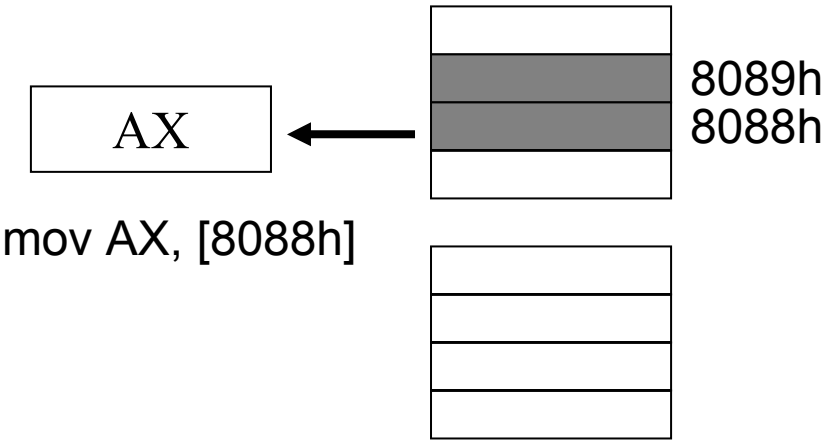
```
mov     ax, bl
mov     al, bx
```

- 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

Esempi di indirizzamento con istruzione mov

- ***Immediate*** `mov ax, 10h`
- ***Direct***

```
mov al, ds:[8088h]
mov ds:[1234h], dl
```

ds si potrebbe omettere: per default un indirizzo diretto fa riferimento al data segment. Se si vuole specificare un offset in un altro segmento si userà il prefisso relativo al segmento (ad esempio: nell'extra segment si userà es)

- ***Register***

```
mov ax, bx    ;il dato da trasferire in ax è in un registro (bx)
```

- ***Register indirect***

```
mov al, [bx]
mov al, [di]
mov al, [si]
```

} Usano il segmento dati (ds) di default

si possono usare espliciti prefissi relativi al segmento se si desidera accedere a dati in segmenti differenti da quelli di default: ad esempio, mov al, ss:[si]

Esempi di indirizzamento con istruzione mov

- ***Register relative***

```
mov     al, 20h[bx]
mov     al, 20h[di]
mov     al, 20h[si]
mov     al, 20h[bp]
```

Usano il segmento dati (ds) di default

Usa il segmento stack (ss) di default

- ***Based indexed***

```
mov     al, [bx][di]
mov     al, [bx][si]
mov     al, [bp][di]
mov     al, [bp][si]
```

Usano il segmento dati (ds) di default

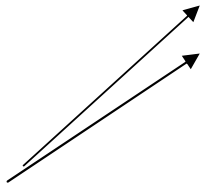
Usano il segmento stack (ss) di default

- ***Relative based indexed***

```
mov     al, 20h[bx][di]
mov     al, 20h[bx+si]
mov     al, [bp+di+20h]
mov     al, 20h[bp][si]
```

Usano il segmento dati (ds) di default

Usano il segmento stack (ss) di default



Possibili variazioni sintattiche

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:

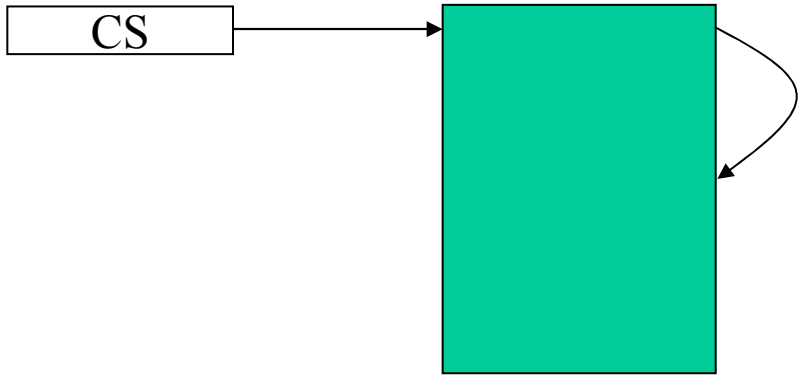
```
mov al, 8[ebx][esi*4]          ; copia in al il dato nella locazione[1] 1018h
mov al, 1000h[ebx][ebx*2]  ; copia in al il dato nella locazione[1] 4000h
mov al, 1000h[esi*8]           ; copia in al il dato nella locazione[1] 1020h
```

[1)] è sempre comunque un effective address

Modi di indirizzamento in caso di salto

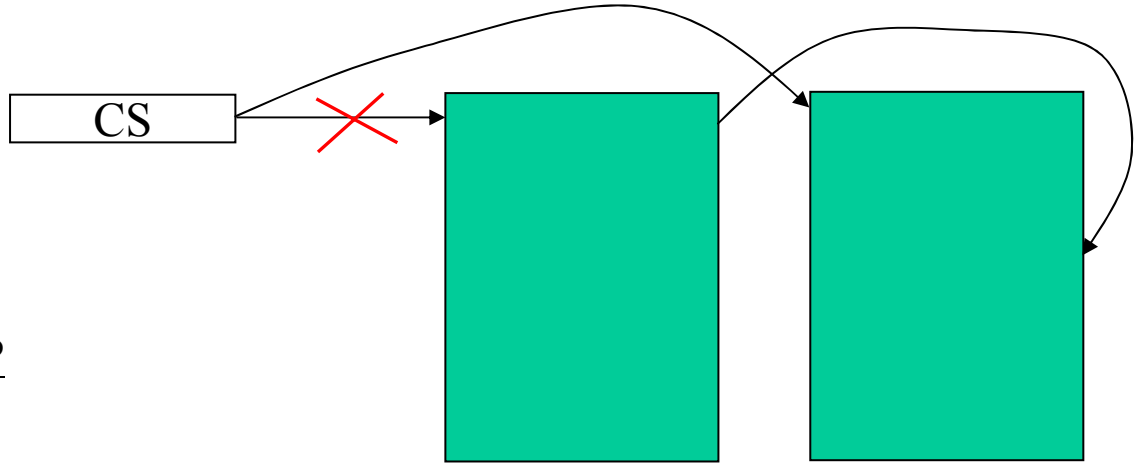
- Dobbiamo specificare un indirizzo fisico a cui saltare

1. Intrasegment



Varia solo IP

2. Intersegment

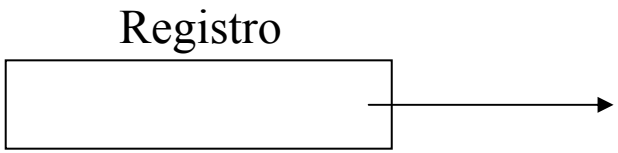


Variano CS e IP

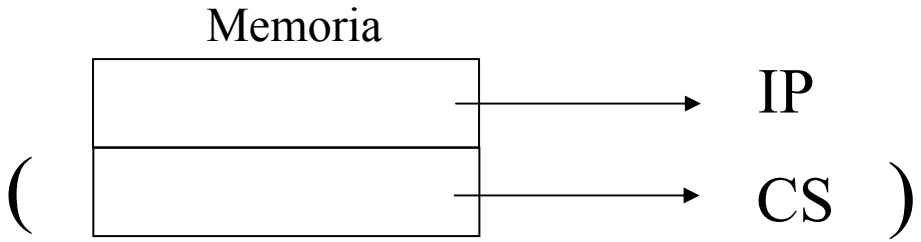
- Come specifico IP [e CS]?

1. Direttamente nell'istruzione (Direct)

2. Con uno dei modi di indirizzamento per i dati (Indirect)



[Register,
solo nell'intrasegment]

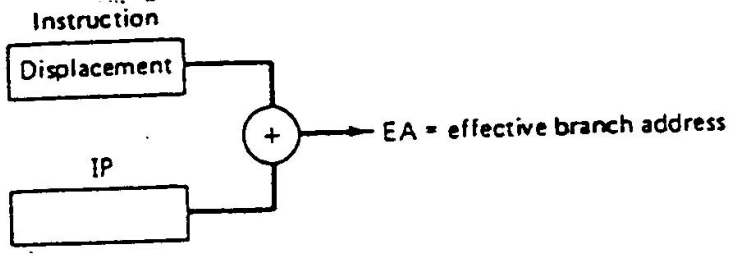


[Diretto, Reg. Ind.,
Reg. Rel., Based Index.
Rel. Based Index.]

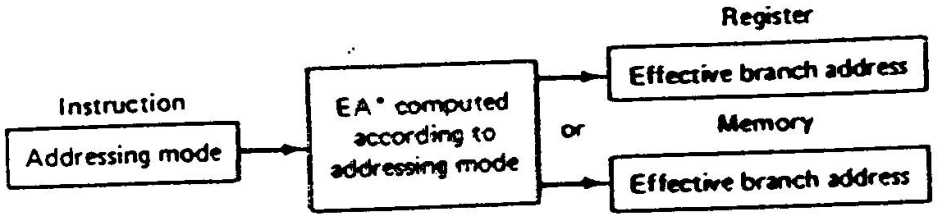
Modi di indirizzamento in caso di salto

- ***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

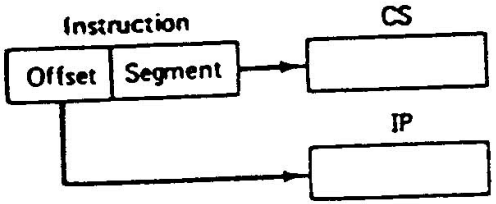
NB: i salti condizionati possono usare solo Intrasegment con scostamento a 8 bit



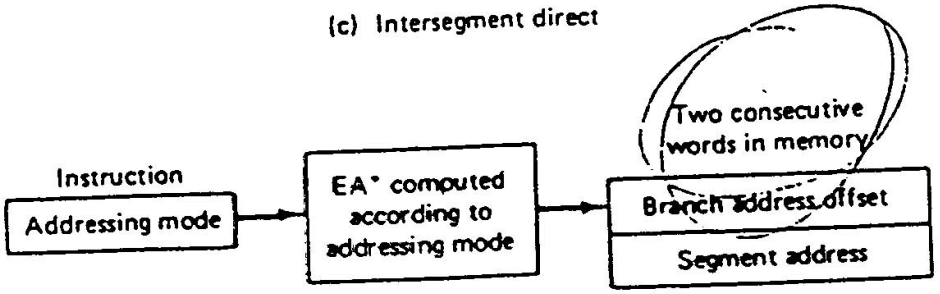
(a) Intrasegment direct



(b) Intrasegment indirect



(c) Intersegment direct



(d) Intersegment indirect

*EA is added to 16_{10} times the contents of the appropriate segment register.

Formato delle Istruzioni

Classi di istruzioni

Il set di istruzioni dell'Assembler Intel 80x86 può essere suddiviso nelle seguenti classi:

- Istruzioni di *trasferimento*: es. mov, push, pop
- Istruzioni *aritmetiche*: es. add, sub, cmp, mul, div
- Istruzioni *logiche*: es. and, or, xor, not
- Istruzioni di *manipolazione di bit*: es. shl, shr
- Istruzioni di *controllo*: es. jmp, call, ret, jg, jge, loop
- Istruzioni di *input/output*: es. in, out
- Istruzioni di *manipolazione di stringhe*: es. movs, stos, lods
- Istruzioni di *controllo dello stato della macchina*: es. hlt, wait

Codifica delle istruzioni

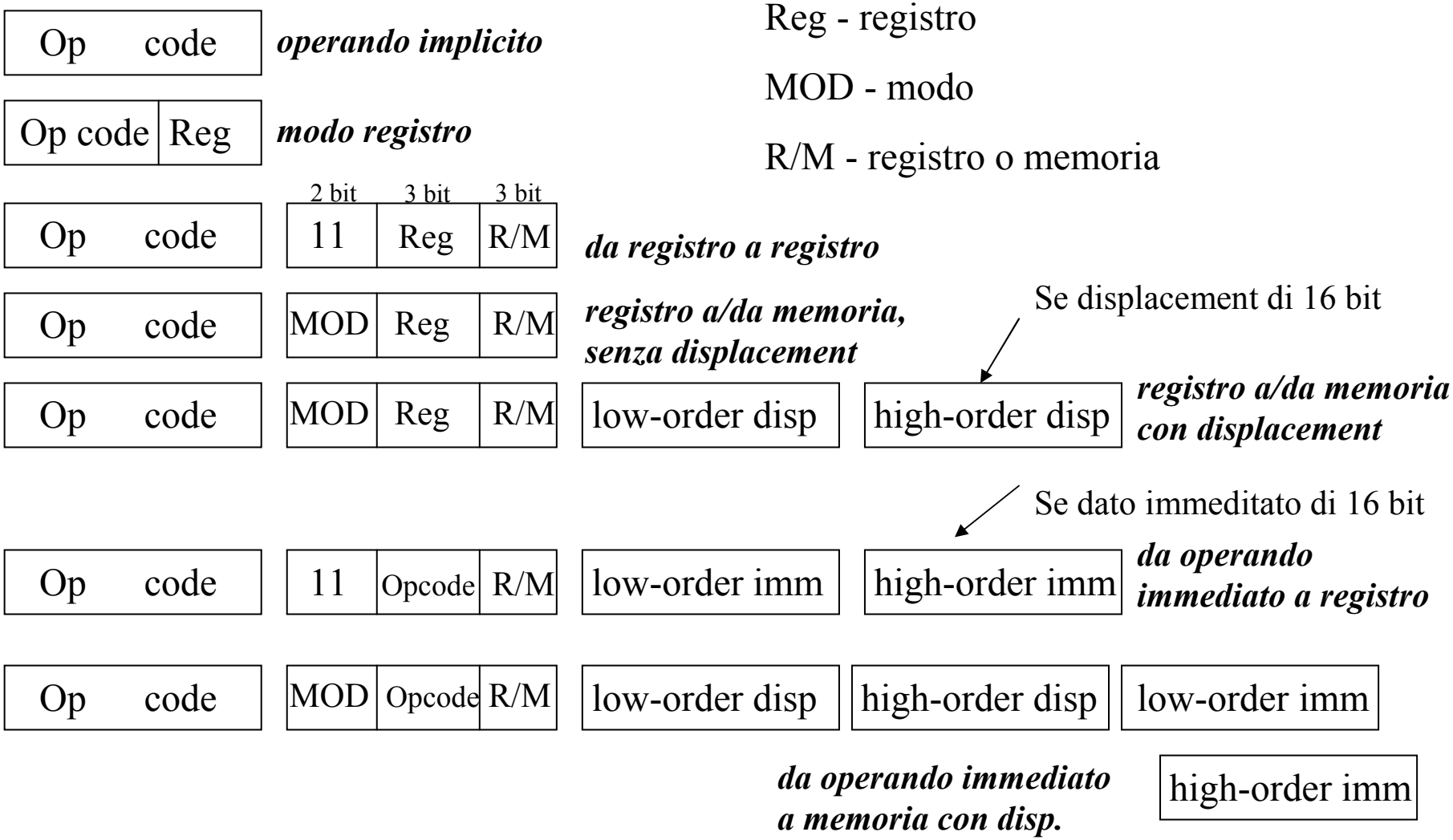
- Codifica delle istruzioni complessa: molti formati diversi
- Nell'8086 le istruzioni possono essere lunghe da 1 a 6 byte
- Nelle architetture successive le istruzioni possono essere lunghe fino a 17 byte
- Un'istruzione è sempre formata da un primo (a volte unico) byte che contiene il *codice operativo*
- 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]
  - 1o 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

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

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”

MOD **R/M**	**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.]

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

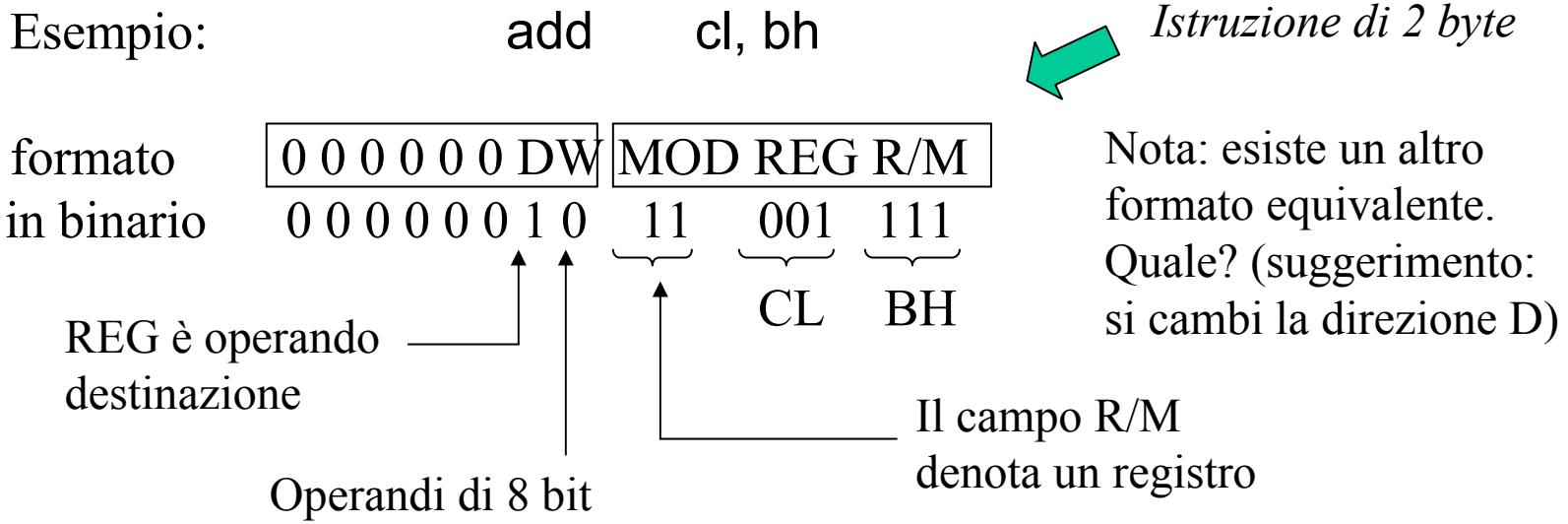
- Esistono comunque delle eccezioni, p.es. CS è sempre il registro segmento nelle istruzioni di salto, SS quando si usa Stack Pointer [istruzioni PUSH e POP]

Esempio: istruzione ADD (1)

- L'instruzione add può avere diverse forme:

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

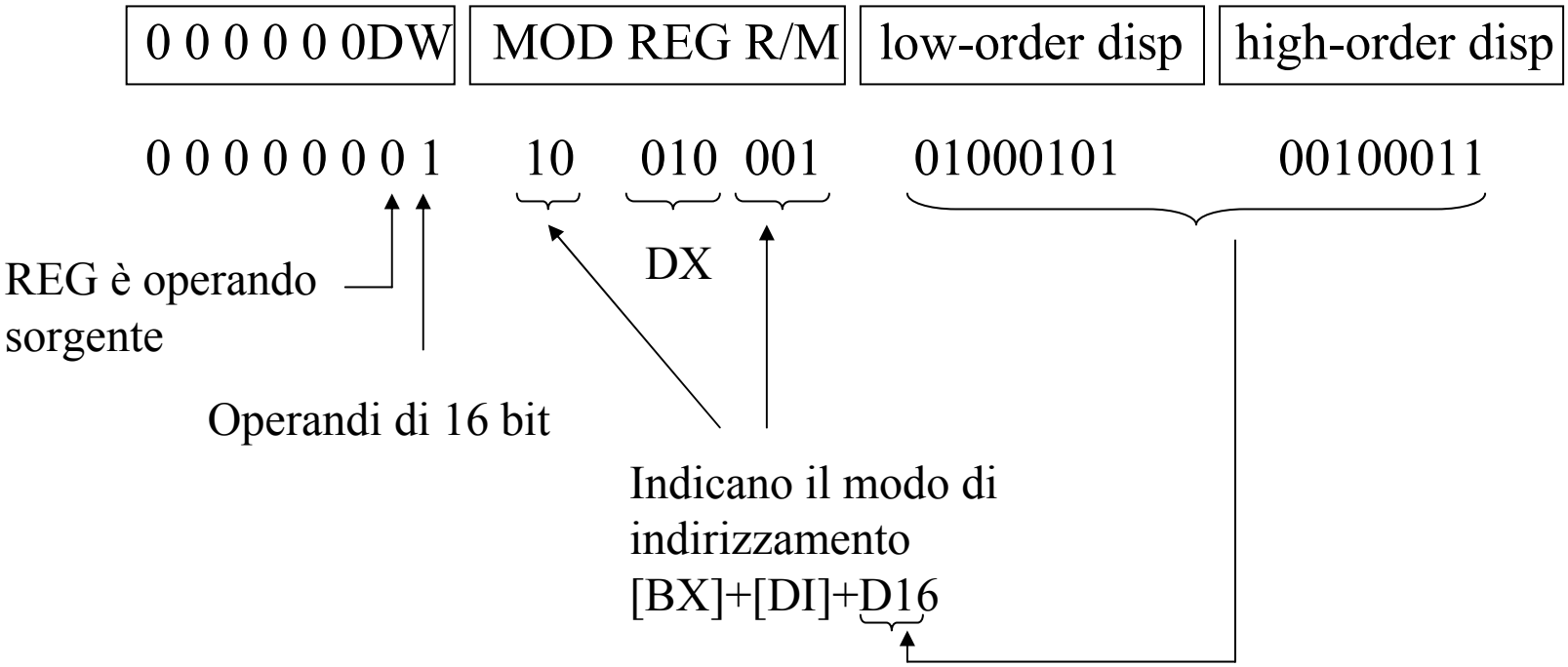
- Tutte aggiungono il secondo operando al primo lasciando il risultato nel primo operando
- Esempio: add cl, bh



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 primo 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)
- La restrizione di cui sopra è legata al numero limitato di registri nell'Intel 80x86

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, 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