# Rodica Baciu

## Programare în limbaj de asamblare

# Bibliografie


1. **BACIU, R.,** *Programarea în limbaje de asamblare – note de curs, probleme rezolvate, probleme propuse*, Ediţia a-IV-a, Editura Techno Media, Sibiu, 2010 (I.S.B.N. 978-606-8030-82-1) (242 pagini). (**Un curs tiparit/grupa**)

2. Athanasiu, I., Pănoiu, A., *Microprocesoarele 8086/ 80286/ 80386*, Editura Teora, Bucureşti, 1992

1. Muscă, Ghe., *Programare în limbaj de asamblare*, Editura Teora, Bucureşti, 1998

2. Lungu, Vasile, *Procesoare INTEL, Programare în limbaj de asamblare*, Editura Teora, 2000

3. Căprariu, V., Enyedi, A., Muntean, M., *Sistemul de operare DOS - Ghidul programatorului*, Editura Microinformatica, Cluj, 1992

4. Manual de utilizare Felix PC

5. Borland C++ vers.4.0, Manuale de firmă

# Informatii utile

- Nota finala = 35%**NL**+65%**NE**
- **NL**=prezenta activa la laborator+nota test scris la mijlocul semestrului+colocviu practice la sfarsitul semestrului
- **NE** examen scris “face to face”

sau

examen on-line (despre care vom mai discuta)

# Informatii utile-continuare

- Prezenta curs: minim 50% din total
- Prezenta laborator: minim 12 laboratoare

# Structura cursului

## *Microprocesorul 8086*

## *LIMBAJ*

- *Setul de instrucţiuni 8086*
- *Directive şi operatori*
- *Macroinstrucţiuni*

## *Utilizarea procedurilor în limbaj de asamblare*

- *Transmiterea parametrilor către proceduri*
- *Întoarcerea datelor de către proceduri*
- *Variabilele locale ale procedurilor*
- *Proceduri recursive şi funcţii recursive*

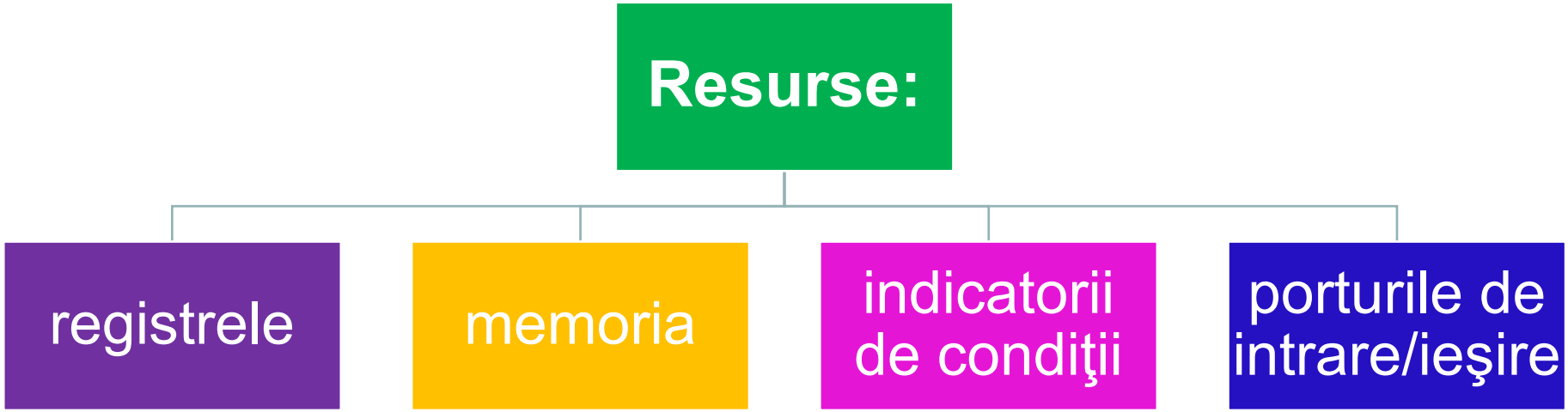
## *Aplicaţii mixte ASM-C*

# *Microprocesorul 8086*

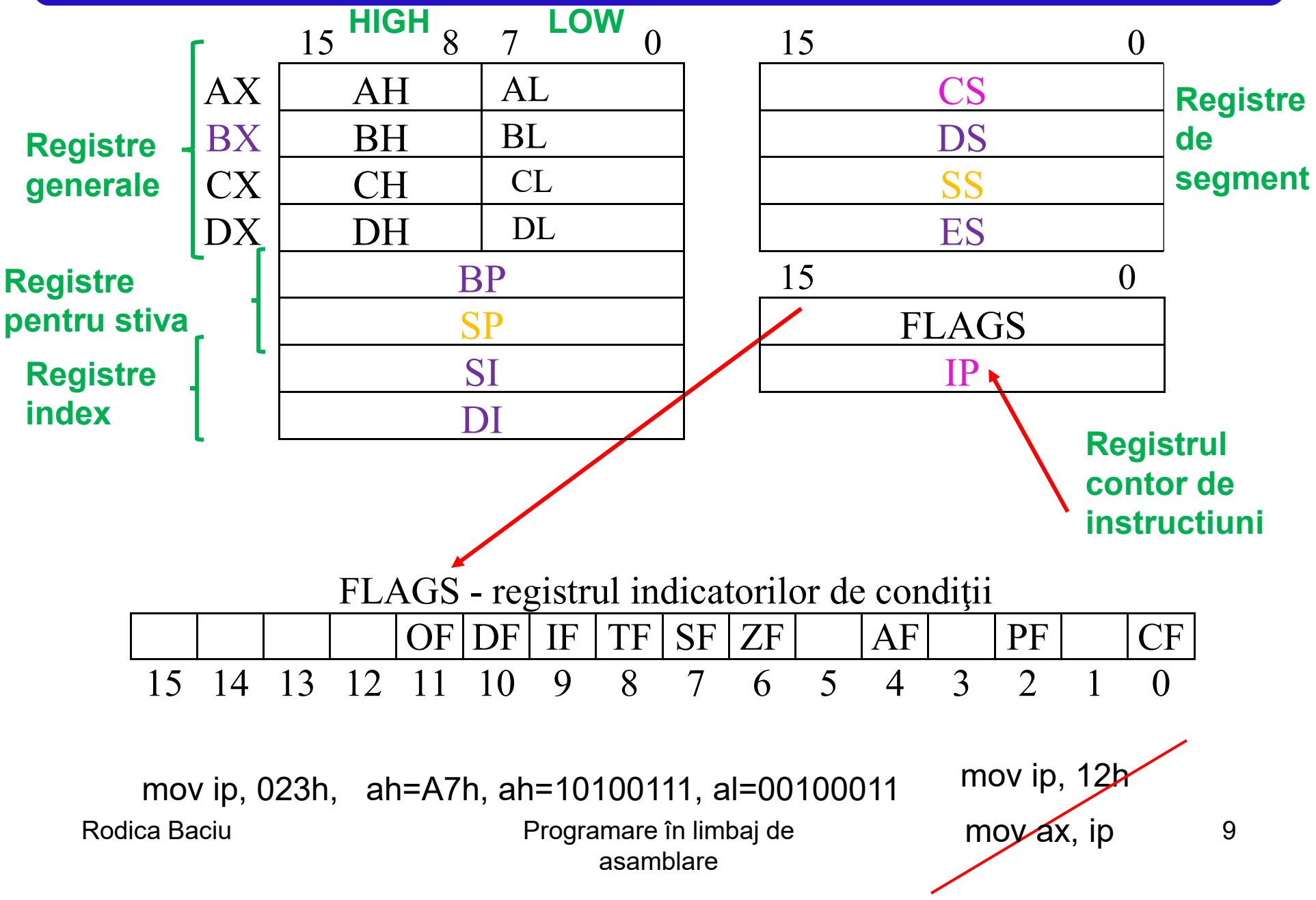
- Registrele microprocesorului 8086
- Memoria (conceptul de segmentare a memoriei)
- Formarea adresei fizice
- Stiva
- Moduri de adresare
- Tipuri de date utilizate în limbaj de asamblare

# Capitolul 1
# *Microprocesorul 8086*

# Resursele instrucțiunilor



# Registrele microprocesorului 8086

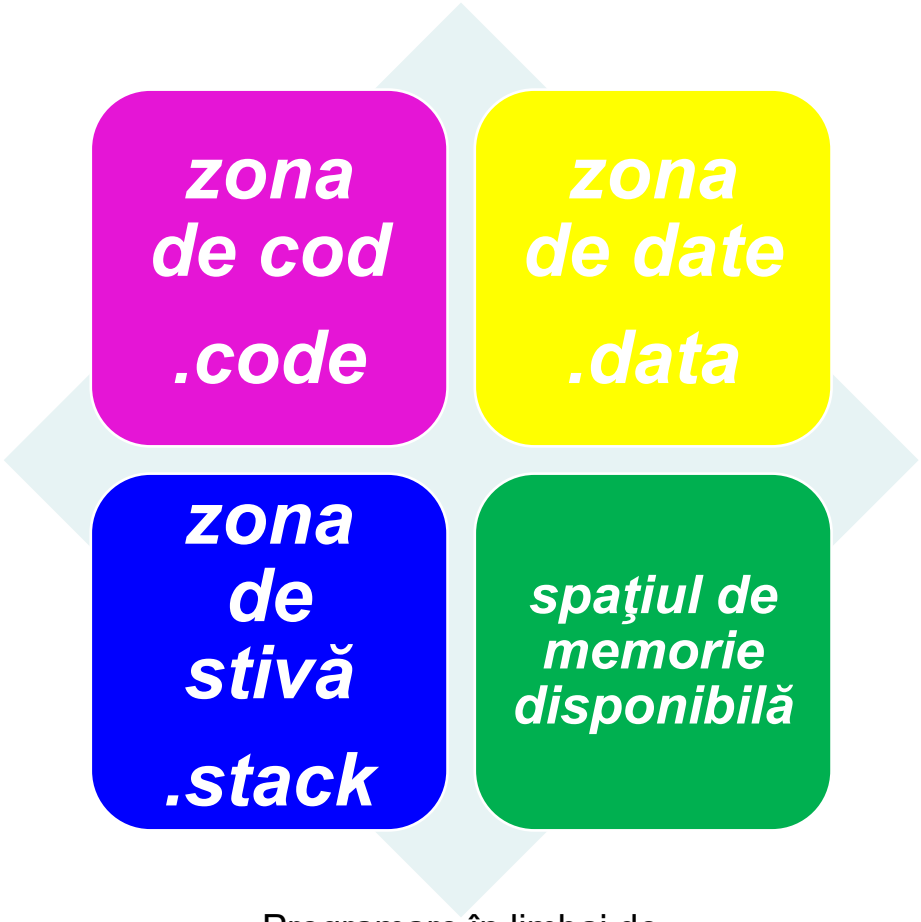



mov ip, 023h, ah=A7h, ah=10100111, al=00100011

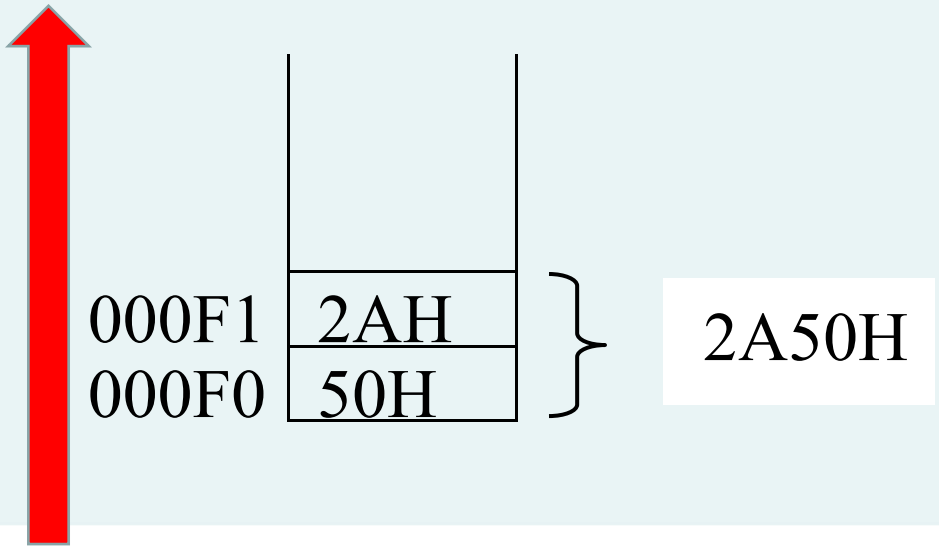
mov ip, 12h

mov ax, ip

# Structura spațiului de memorie alocată unui program executabil



# *Convenţia INTEL:* octetul mai semnificativ se află memorat la locaţia de memorie cu adresa cea mai mare:




```
.data
VAR1 DW 2A50H, 1h
VAR2 DB 50H, 2AH
```

```
mov ax, var1
mov al, var2
mov al, var2+1
```

# Memoria (conceptul de segmentare a memoriei)


A0 - A19

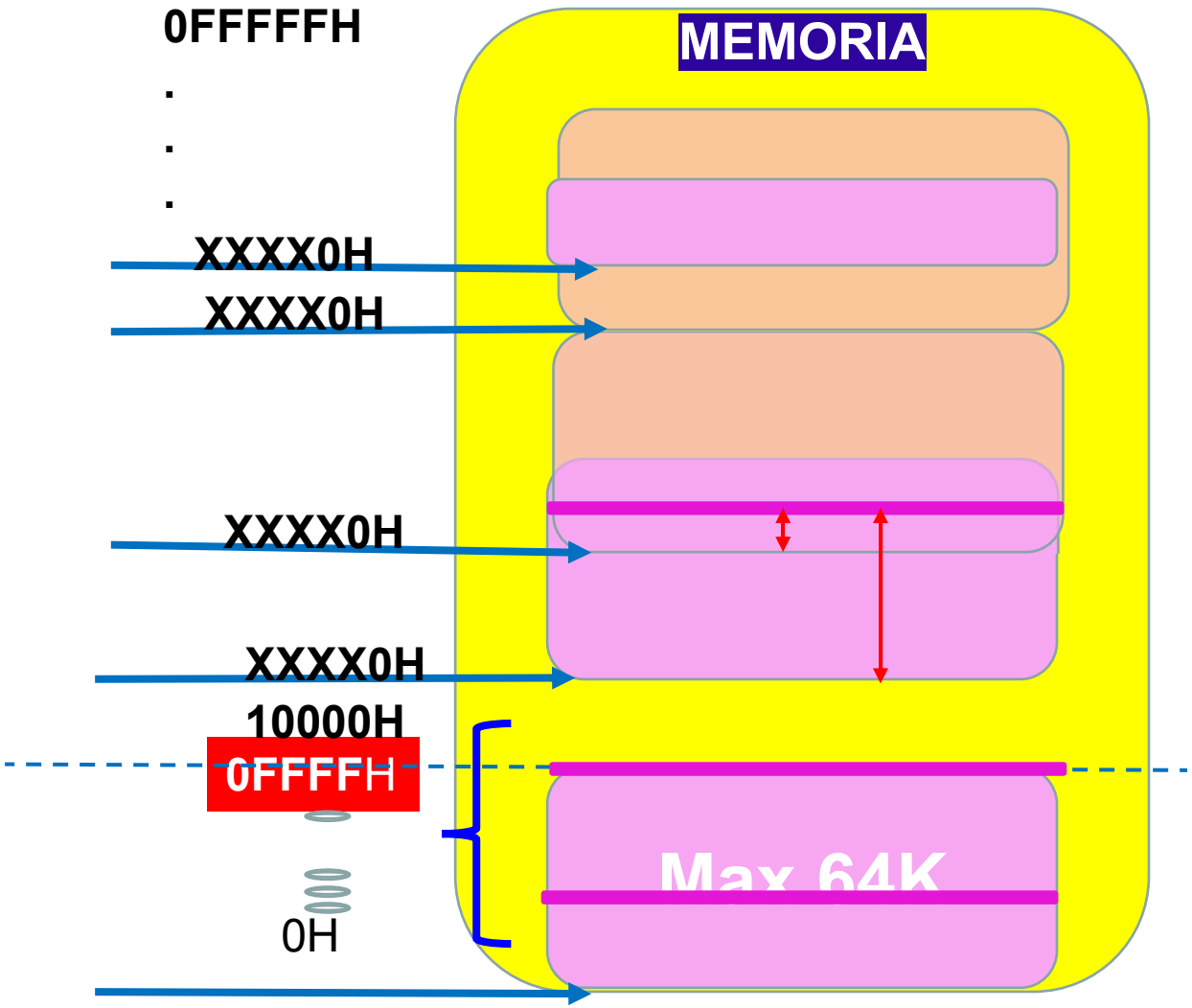
$2^{20}$ -1= 0FFFFFH

0H

MEMORIA

**Capacitatea maxima a memoriei ce poate fi gestionată de 8086: $2^{20} = 2^{10} \times 2^{10}$**
**$\approx 10^3 \times 10^3 = 10^6 = 1$Mo**

# Memoria (conceptul de segmentare a memoriei)

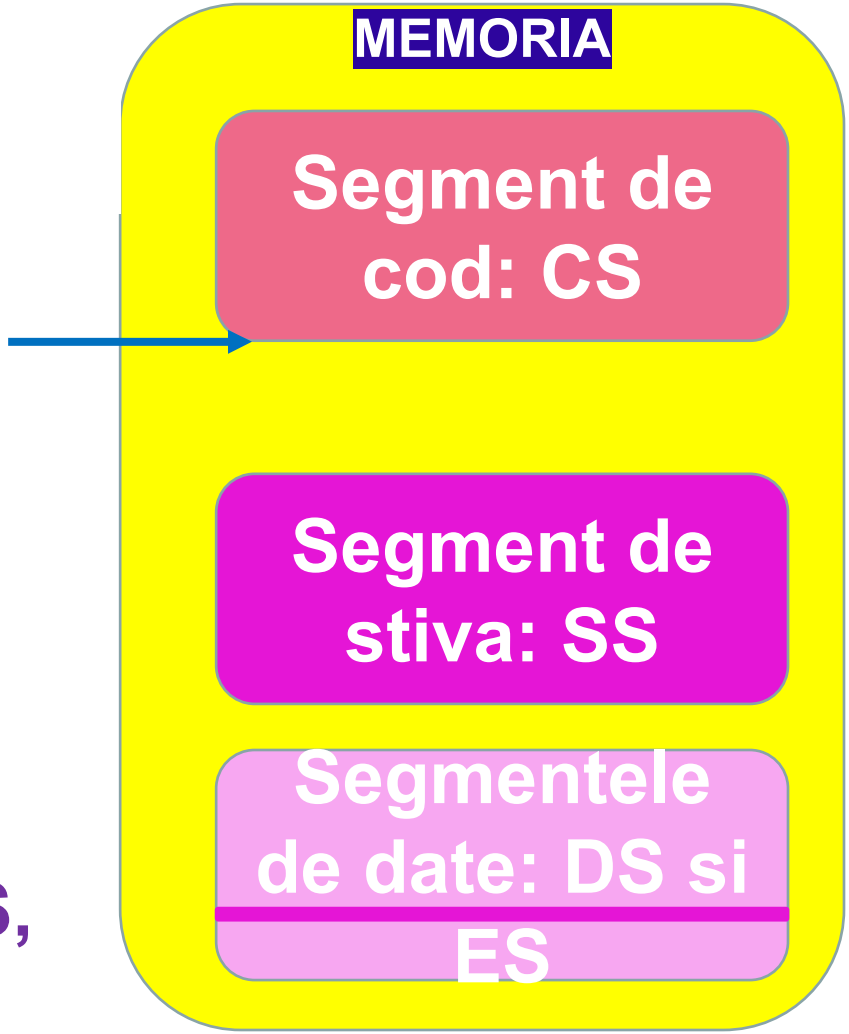


# Segmentele unui program executabil

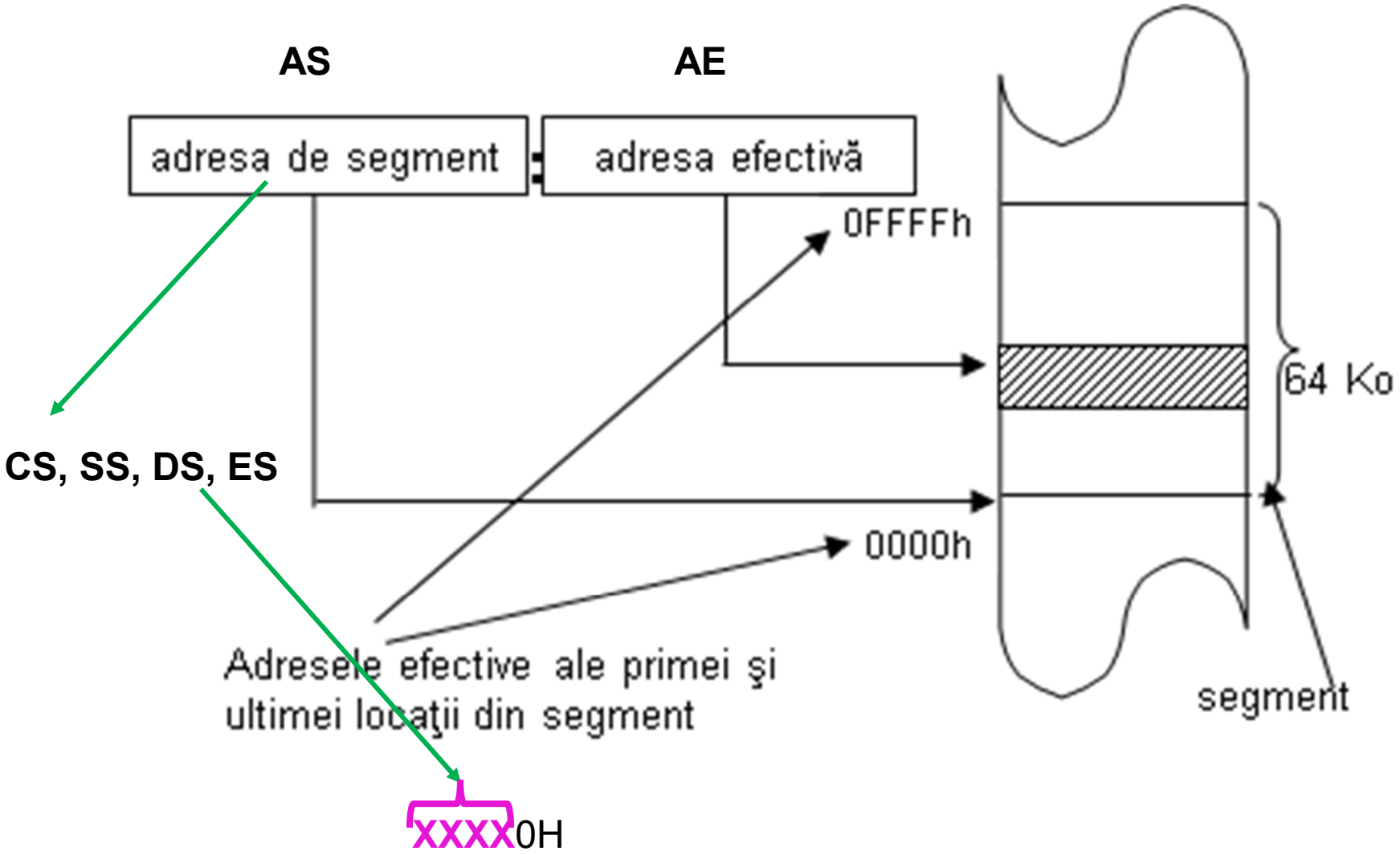
**Locul in memoria principala pentru fiecare segment: SO**

**Cum scrie programatorul adresele datelor cu care lucreaza: AS nu o scrie, AE o scrie pentru date**

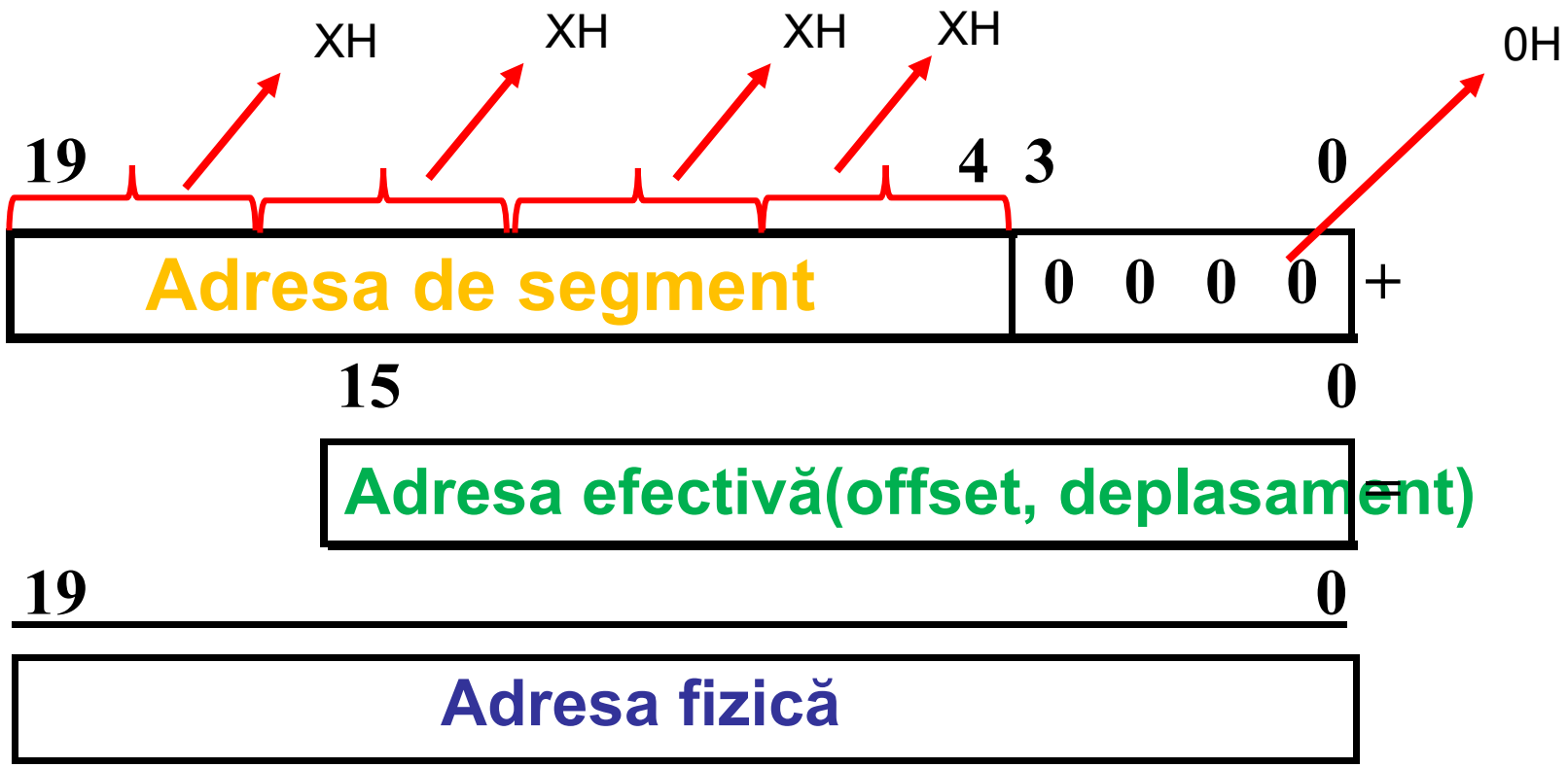
**Care sunt registrele utilizate pentru formarea adresei fiecărui segment: CS, SS, DS, ES**



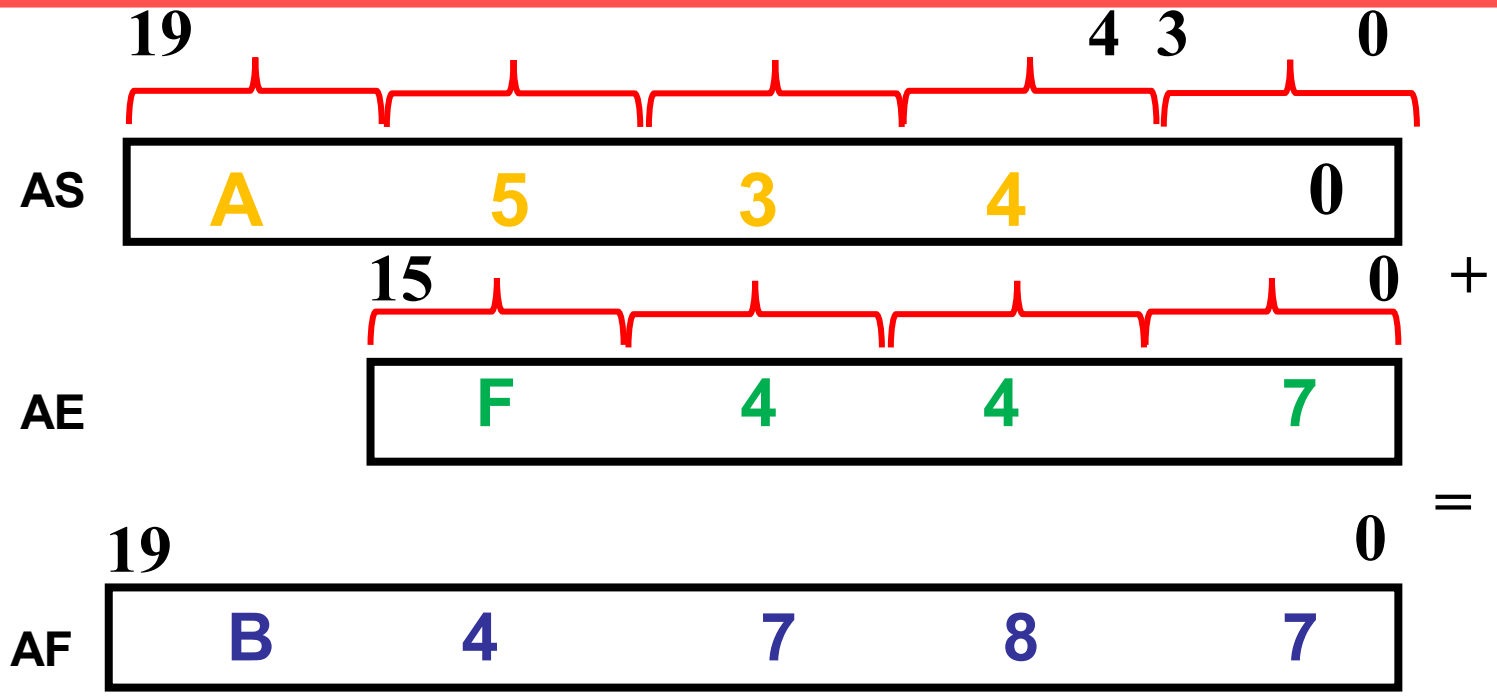
# Formarea adresei fizice a unei locatii de memorie



# Formarea adresei fizice a unei locatii de memorie



# Formarea adresei fizice a unei locatii de memorie. Exemplu




**Adresa de segment** este 0A534h. **Adresa efectivă** este 0F447h.
Adresa fizică se obţine:

```
  0A5340h+
   0F447h
  0B4787h
```


- **IP pt. segmental de cod**
- **SP pt. segmental de stivă**
- **BX, BP, SI, DI pt. segmentele de date**

# Specificarea unei adrese fizice a unei locatii de memorie

**adresă de segment : adresa efectiva(offset)**

**18A3 : 5B27**

sau

**registru segment : adresa efectiva(offset)**

**DS : 5B27**

**Exemple:**

```
mov DX, DS:[BX]
mov DX, [BX]
mov DX, SS:[BX]
mov DX, [BP]
mov DX, SS:[BP]
mov DX, DS:[BP]
```

# Specificarea unei adrese fizice a unei locatii de memorie

**adresă de segment : adresa efectiva(offset)**

**18A3 : 5B27 = 18A30+**
**5B27**
**1E557**

**18A2 : 5B37 = 18A20+**
**5B37**
**1E557**

**18A0 : 5B57 = 18A00+**
**5B57**
**1E557**

# Adrese normalizate

**adresă de segment : adresa efectiva(offset)**

**0.....F**

**18A3 : 0B = 18A30+**

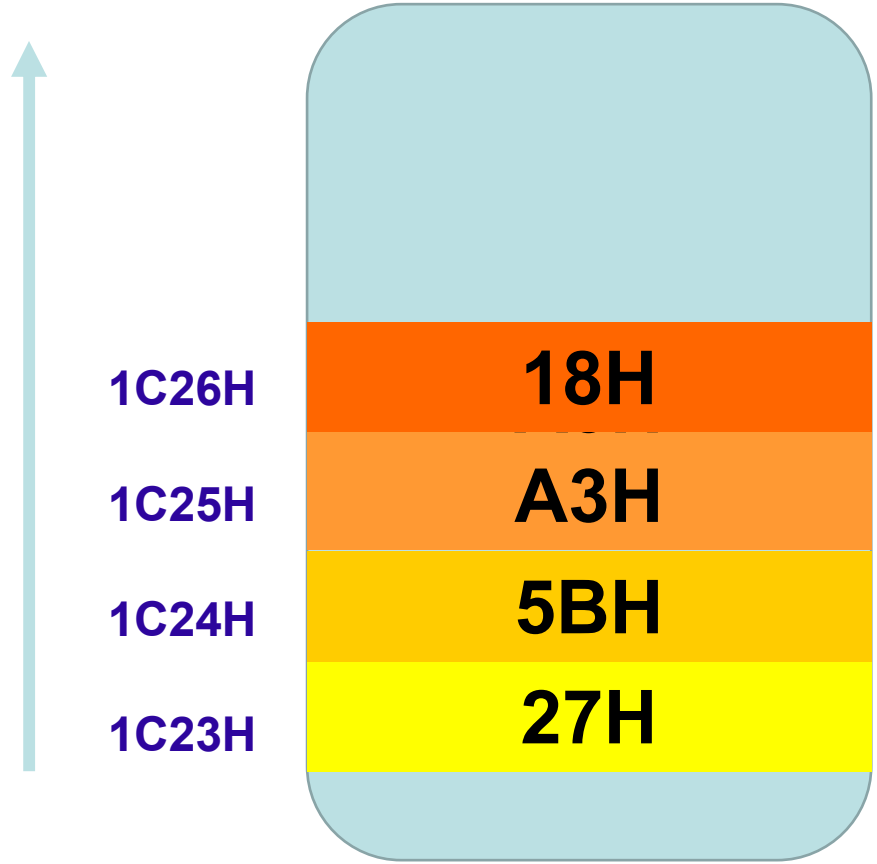
**B**

**18A3B**

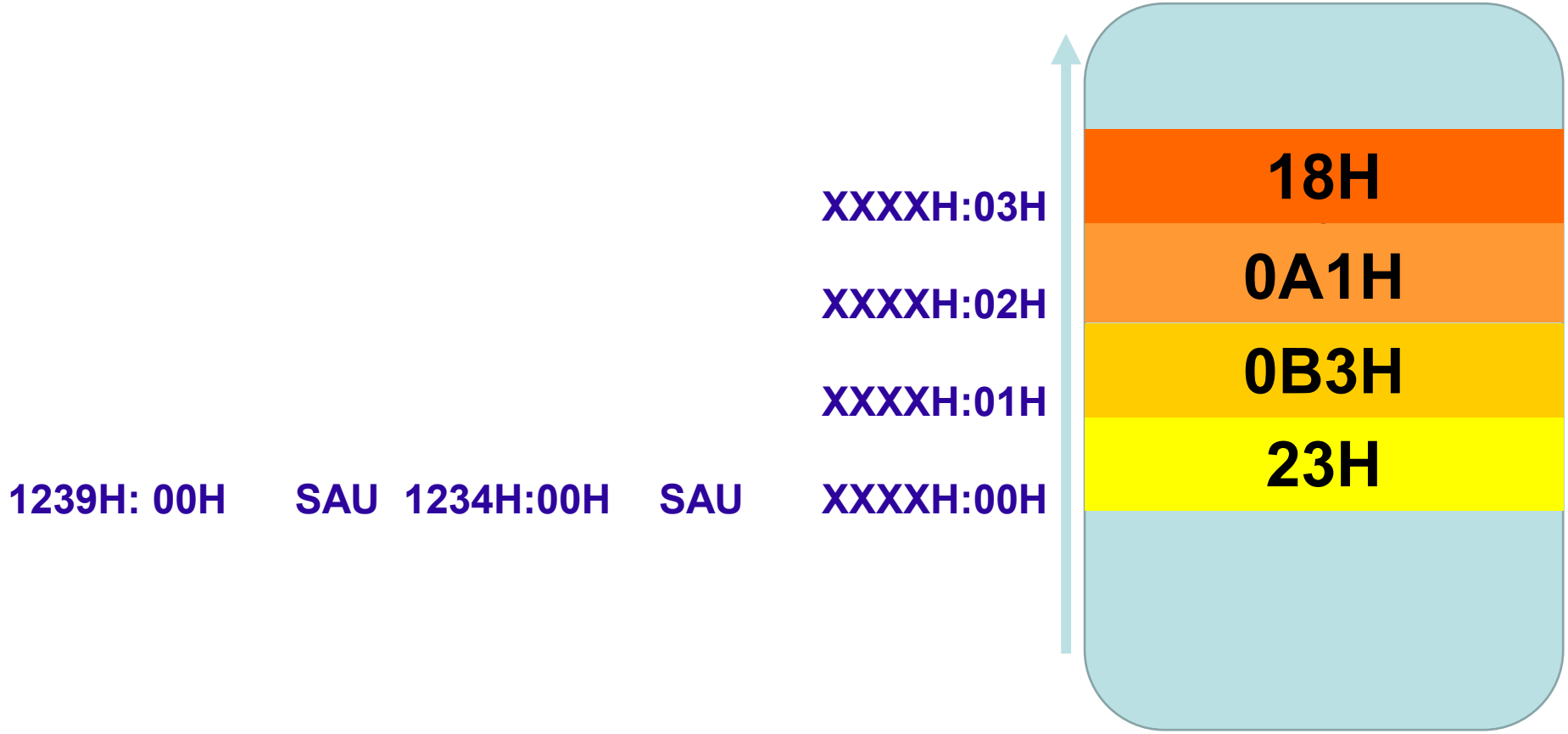
**SUNT UNICE!!!!!!!!!!!!!!!!!!**

# Salvarea în memorie a unei adrese (pointer) respectă *convenţia Intel*:

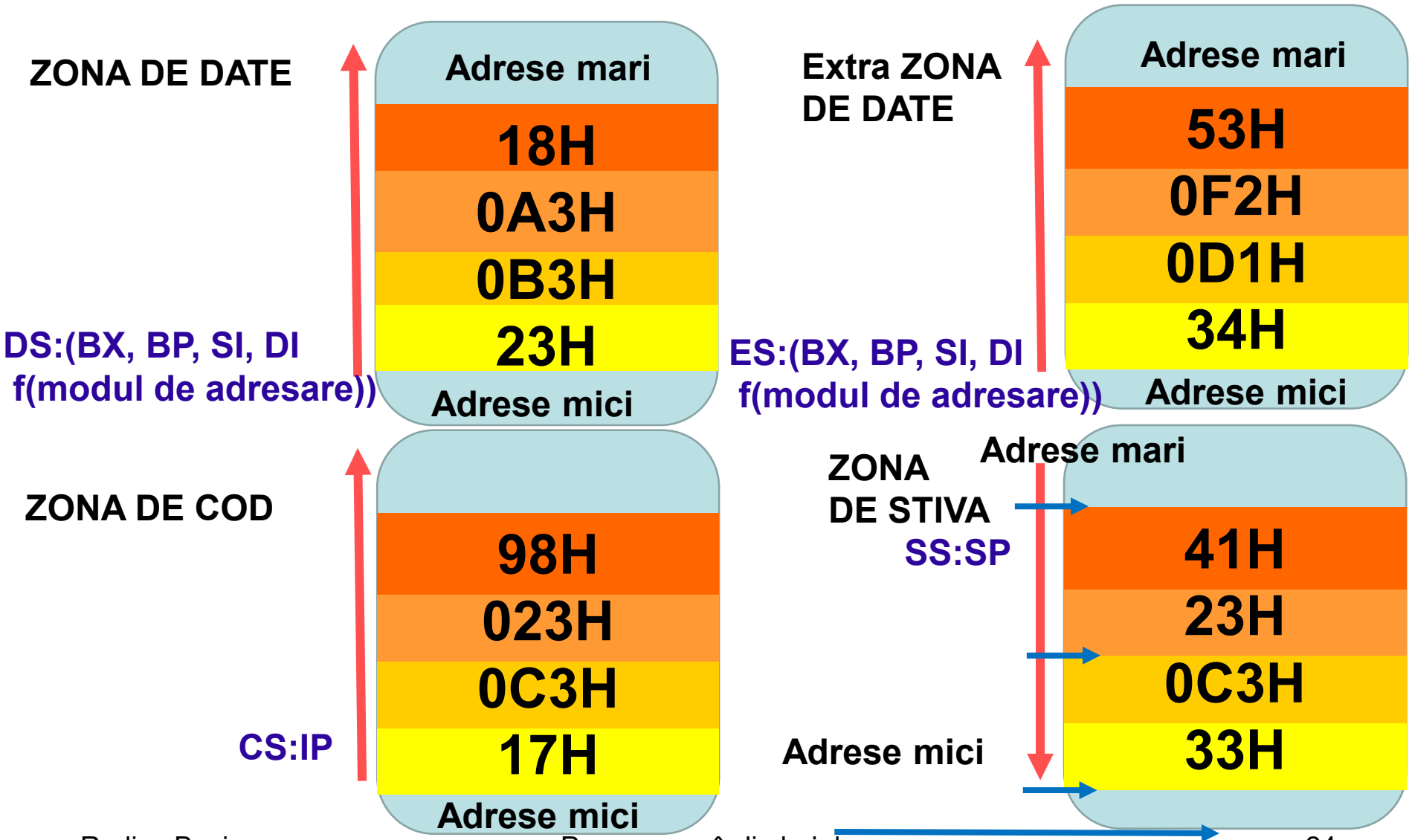
**18A3 : 5B27 se salveaza la adresa 1C23H**



# COD RELOCABIL DINAMIC



# ADRESAREA DIFERITELOR ZONE ALE UNUI PROGRAM EXECUTABIL



# STIVA

- **LIFO**
- **Dimensiunea**
  - Max: 64ko
  - Conform cu DIRECTIVA .STACK
    - Directiva: .stack dim,

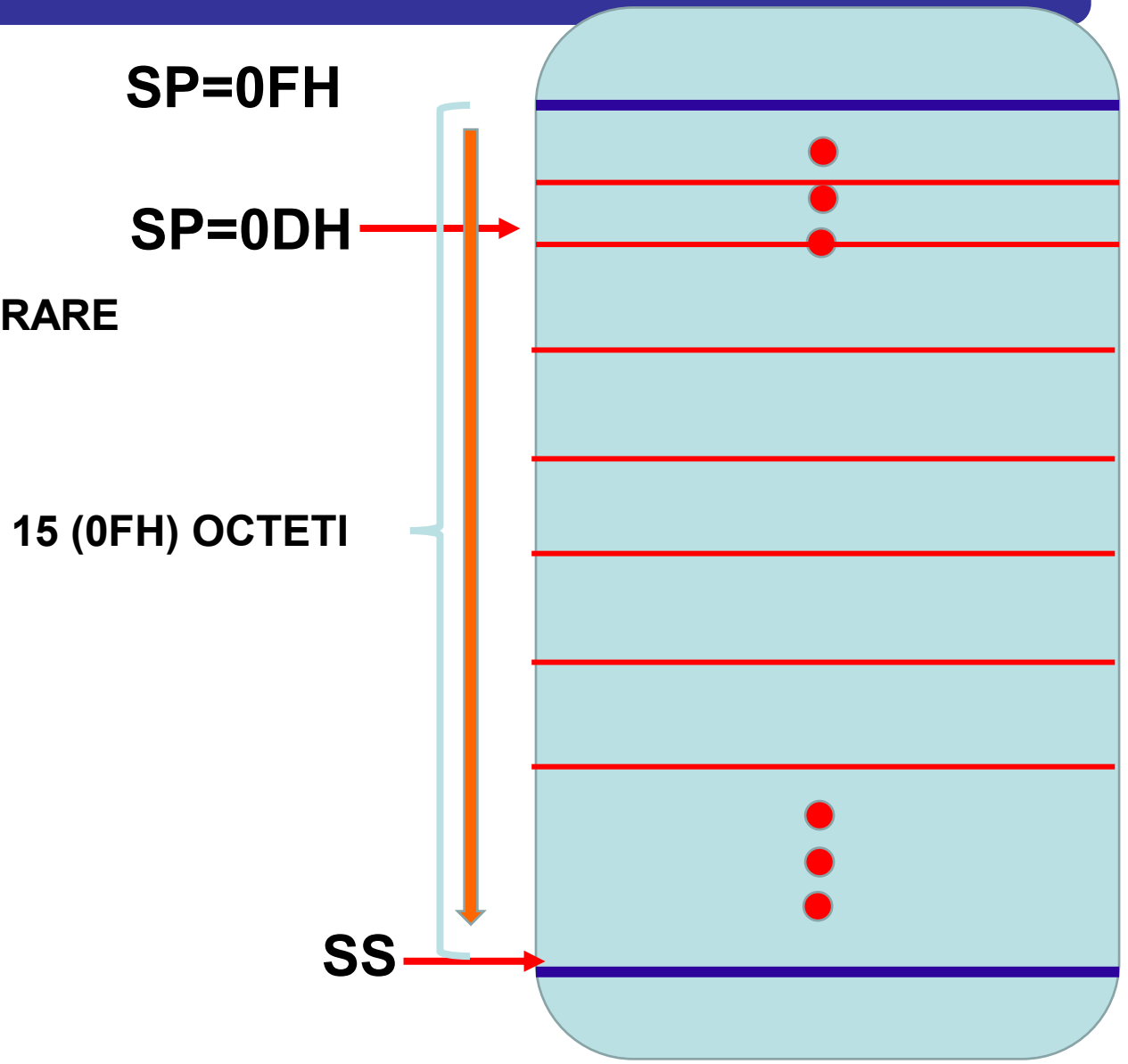
      .stack 100
    - Implicit: 512 octeti

      .stack
    - Min: nu folosim directiva .stack

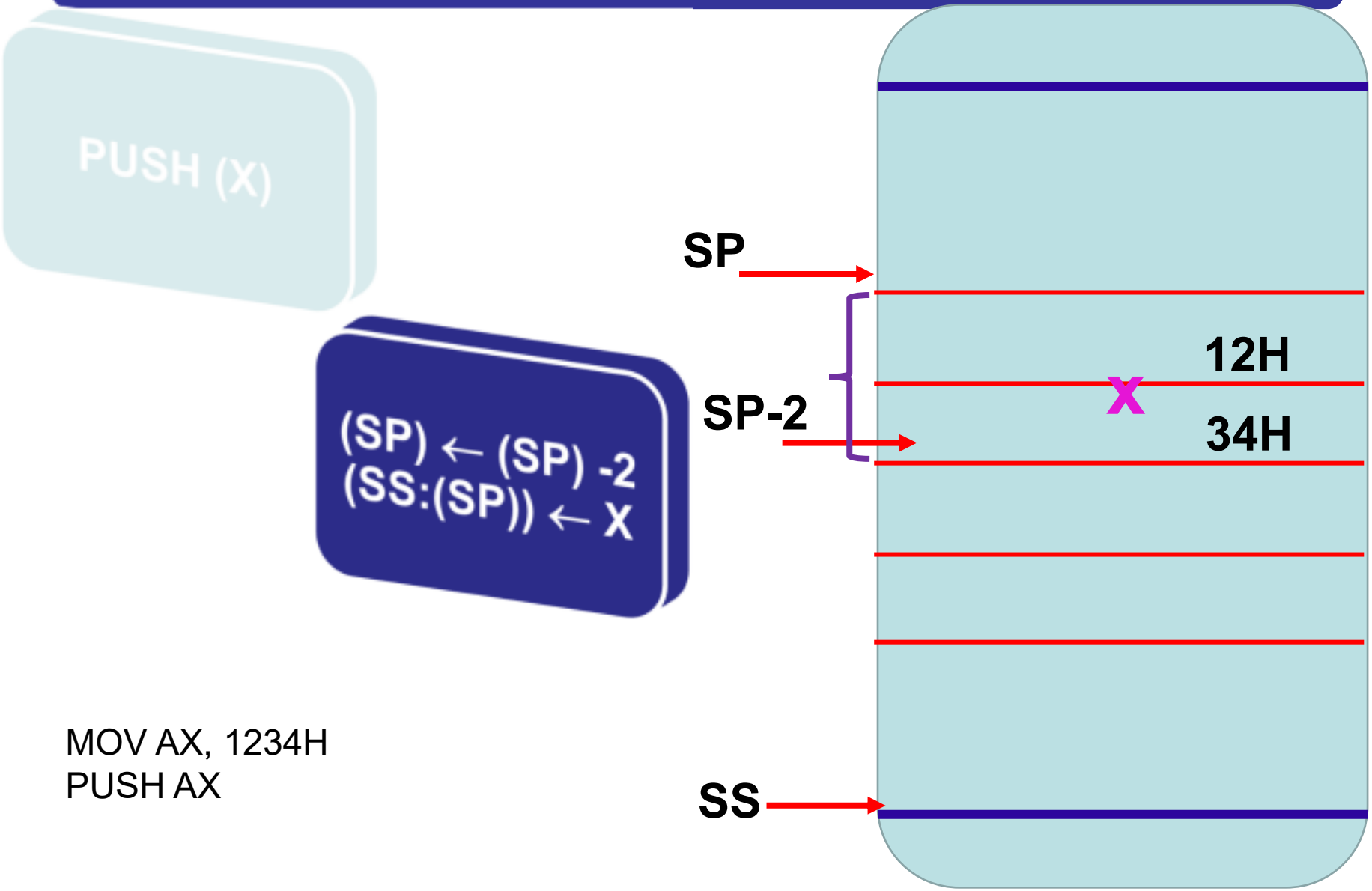
# ADRESAREA STIVEI

**.STACK 15**

**INITIAL:**
**SS ← SISTEMUL DE OPERARE**
**SP ← 0FH**

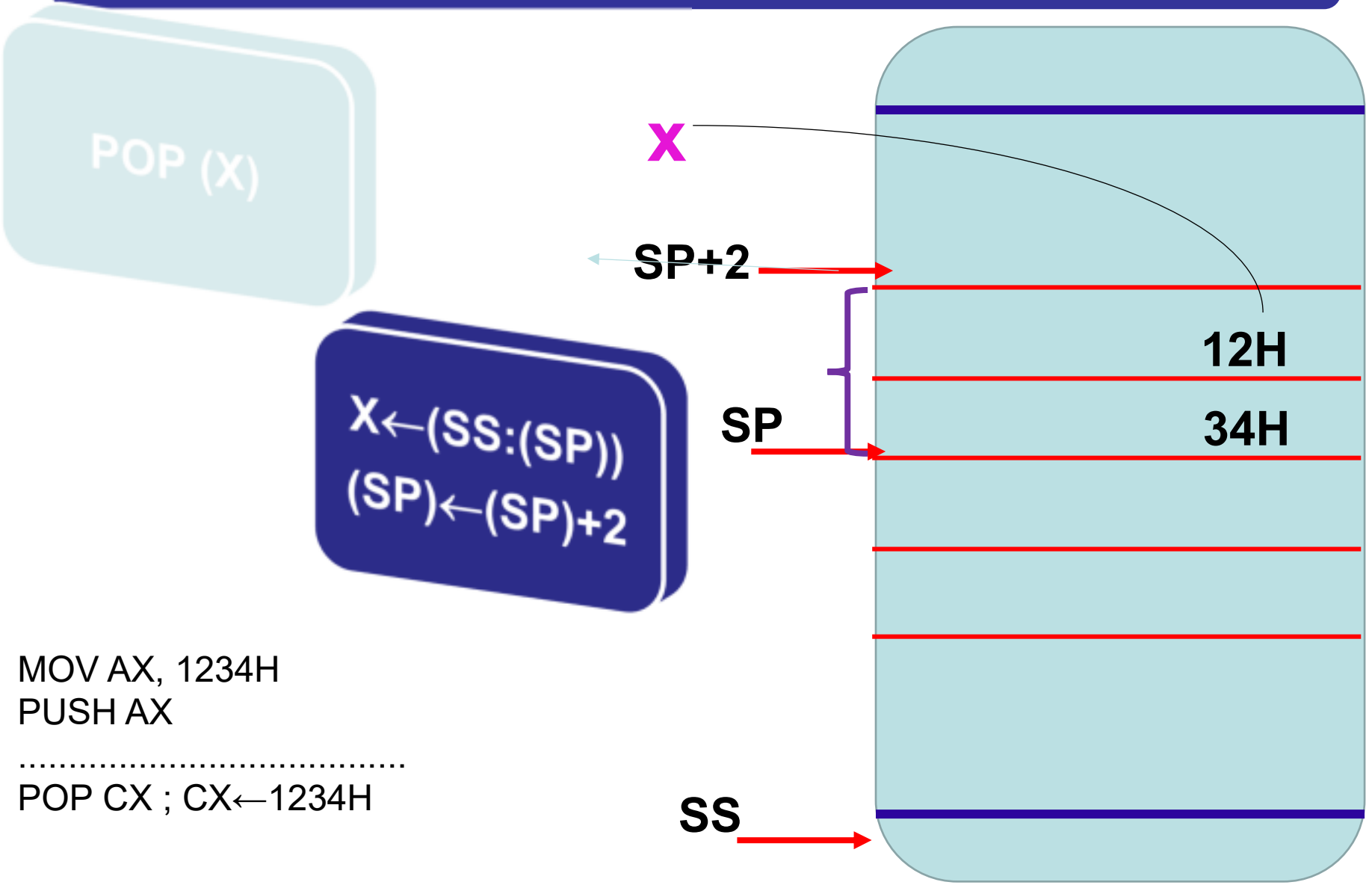


# Operatii EXPLICITE cu STIVA - PUSH



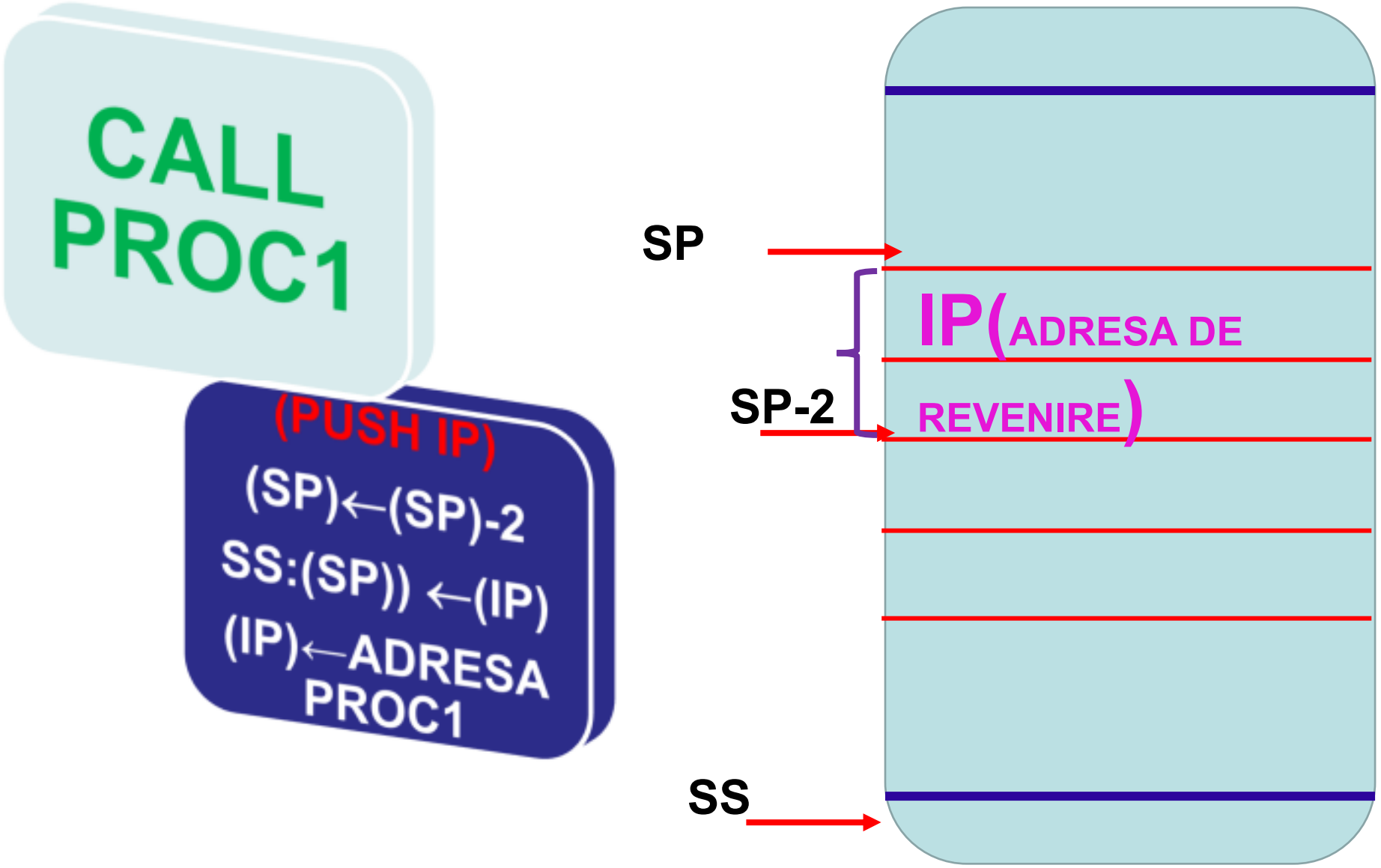

MOV AX, 1234H
PUSH AX

# Operatii EXPLICITE cu STIVA - POP

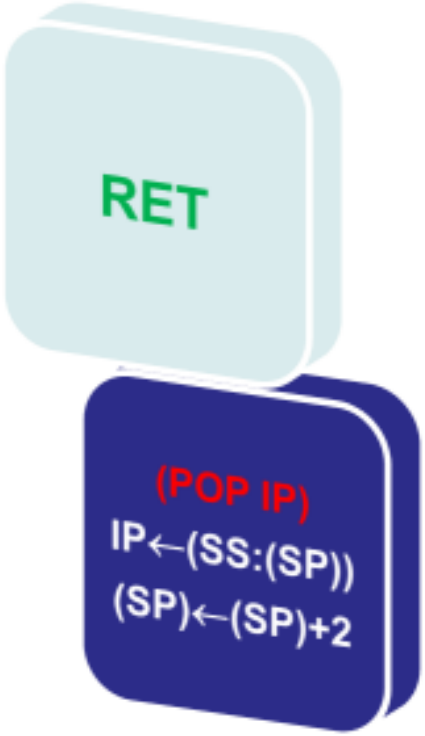


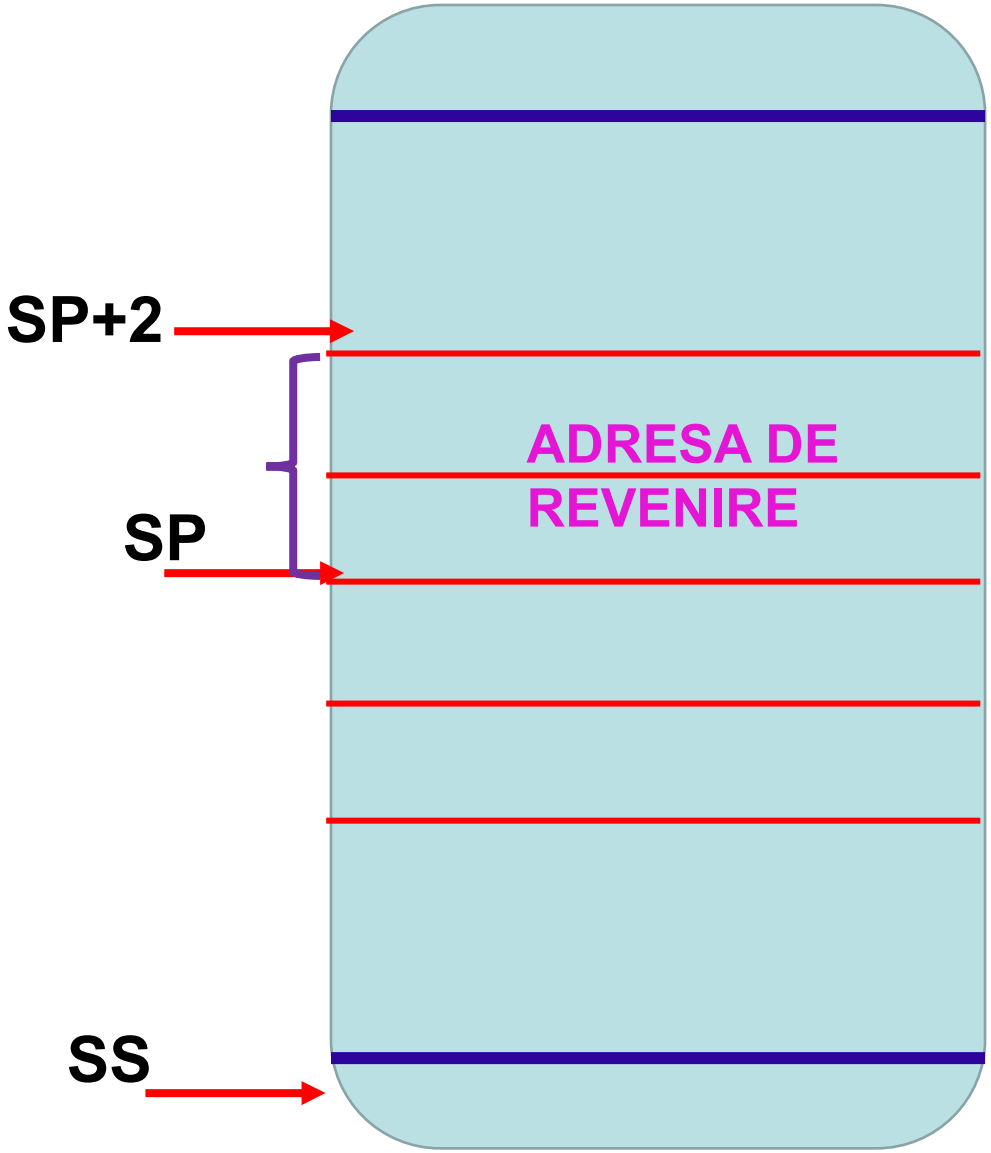

```
MOV AX, 1234H
PUSH AX
.....
POP CX ; CX←1234H
```

# OPERATII IMPLICITE CU STIVA- CALL SI RET

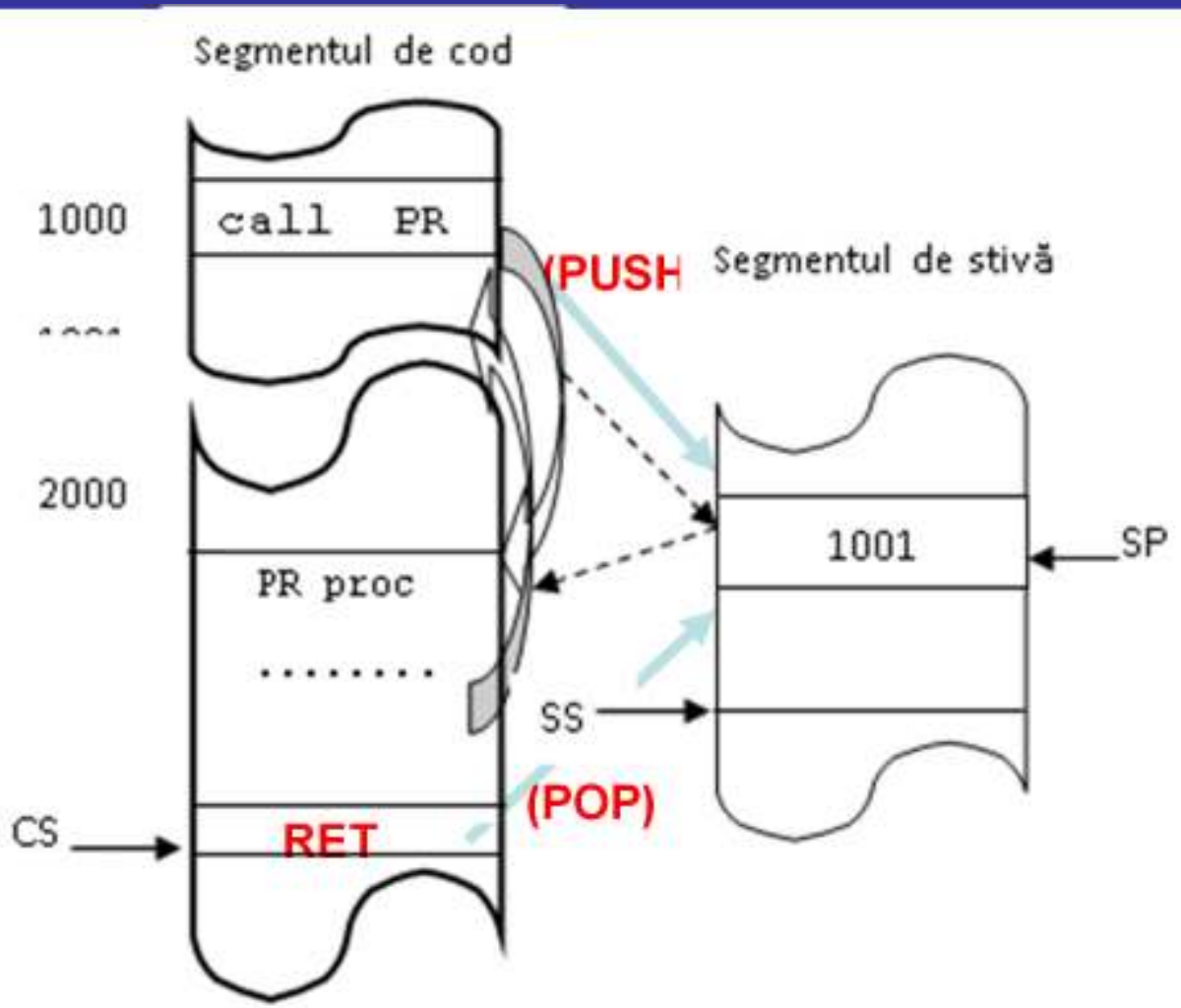


# Operatii IMPLICITE cu STIVA – CALL SI RET



# Operatii IMPLICITE cu STIVA – CALL SI RET



# ACCESAREA STIVEI ALTFEL DECAT FOLOSIND PUSH/POP

FOLOSIND MODURILE DE ADRESARE:


ASTFEL ÎNCÂT SĂ SE LUCREZE IN SEGMENTUL DE STIVĂ (ADRESAT DE SS)

# Registrul indicatorilor de condiţii


| 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 |

# Porturile de intrare/iesire

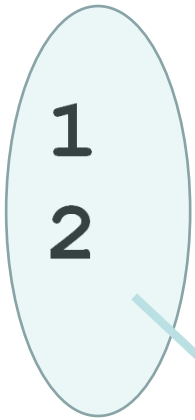
- Registre care NU se află în microprocesor
- Ci in circuitele specializate de pe placa de baza (circuitul de ceas, controlor de intreruperi, port paralel...)

# Moduri de adresare

# Adresarea imediată

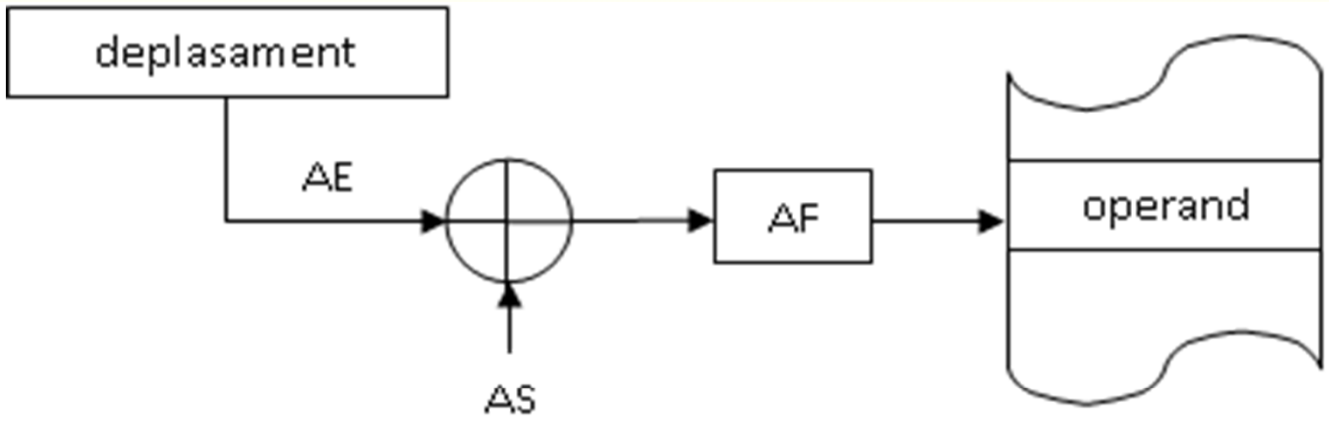
**SINGURUL MOD DE ADRESARE CARE NU LOCALIZEAZA MEMORIA CI SE REFERA LA O VALOARE CONTINUTA DE INSTRUCTIUNE!!!!!!!!!!!!!!!!!!!!**

```
mov     ax, 1     ; pune în AX valoarea 1
add     bx, 2     ; adună la BX valoarea 2
```



**VALORI IMEDIATE**

# Adresarea directă:




AE – adresa efectivă

AS – adresa de segment

Segmentul de date adresat în mod implicit de registrul DS

```
.stack
.data
        VAL   dw   1
        VAL1  DB   23H
.code
        .....
 mov bx, VAL     ;pune în registrul bx valoarea 1   DS:0
 mov bx, es: [100]   ES:100
add   cx, [100]  ;adună la registrul cx ceea ce se găseşte în memorie,   DS:100
                 ;în segmentul de date la offsetul 100.
```

# Adresarea directă:

```
.code
        .....
  mov bx, CS:[100]    ;pune în registrul bx valoarea AFLATA IN SEGMENTUL DE
                      COD LA ADRESA EFECTIVA 100  CS:100
  add  cx, SS:[100]  ;adună la registrul cx ceea ce se găseşte în STIVA,
                      ;LA ADRESA EFECTIVA 100  SS:100
```

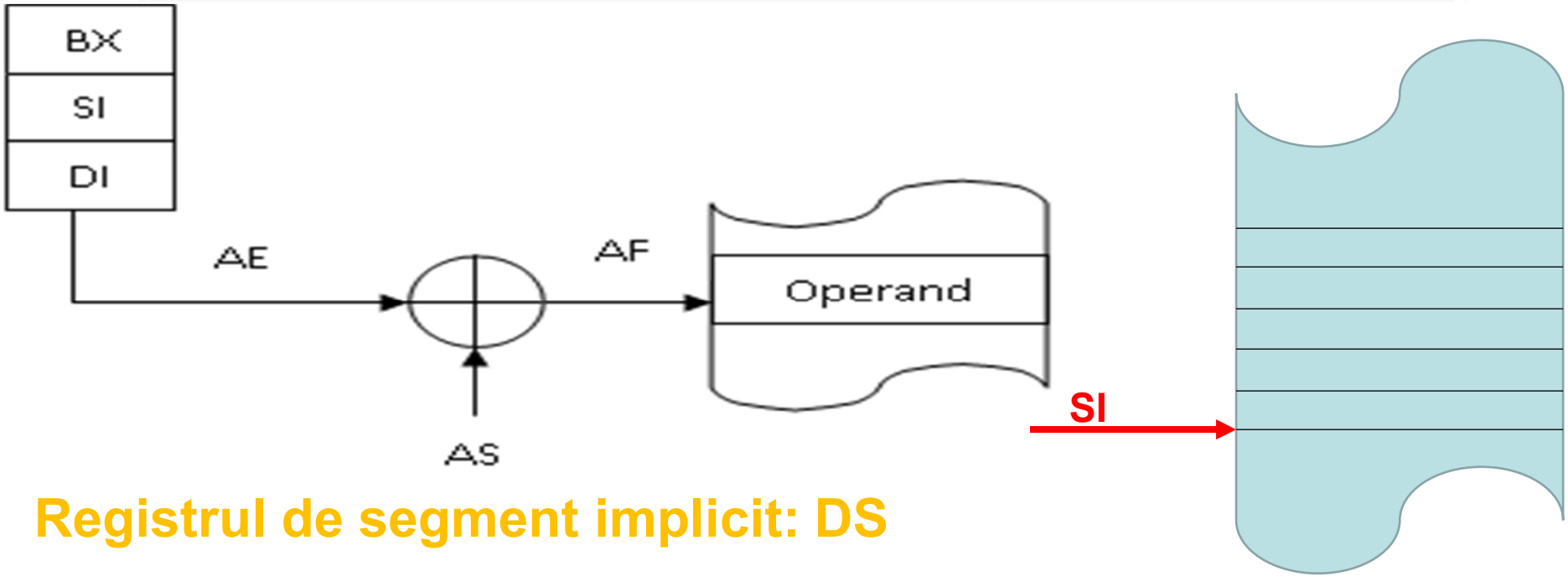
**ESTE O SCOATERE A UNEI DATE DIN STIVĂ, ALTFEL DECAT FOLOSIND POP**

```
  MOV   DX, [100]      ;PUNE IN registrul DX ceea ce se găseşte în SEGMENTUL
                       ;DE DATE LA ADRESA EFECTIVA 100
```

## Exemplu:

Daca DS=1A34H atunci ceea ce se afla la **adresa fizică** 1A340H+100H=1A440H se va pune in registrul DX

# Adresarea indirectă:




## Registrul de segment implicit: DS

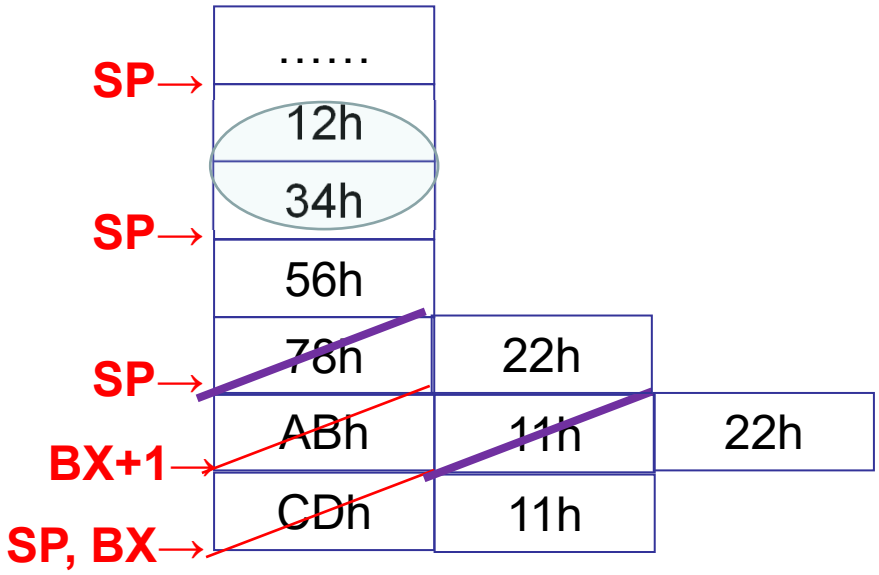
```
mov ax, [bx]          ;conţinutul locaţiei de memorie din segmentul adresat de
                      ;registrul ds de la adresa efectivă specificată în BX
mov [di], cx          ;memorează conţinutul lui CX in segmental de date la
                      ;adresa de offset dată de conţinutul lui DI
add byte ptr ptr[SI], 2     ;adună valoarea 2 la
                      ;octetul aflat in segmental de date la  adresa de offset dată de SI
;operatorul byte ptr este absolut necesar altfel nu s-ar ;cunoaşte tipul operandului
      (octet, cuvânt) la care se adună 2.
```

# Adresarea indirectă:

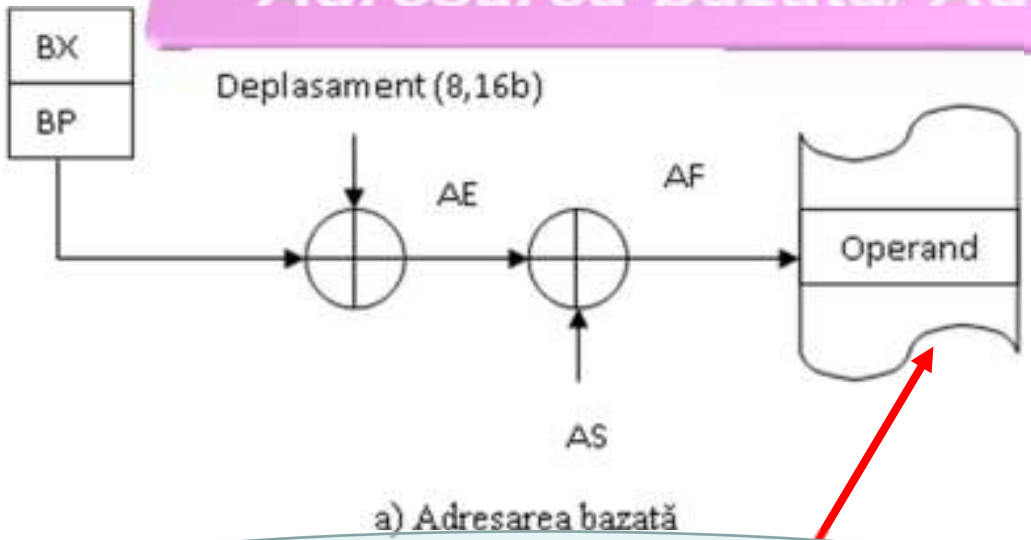
```
mov ax, ss:[bx]      ;conţinutul locaţiei de memorie din segmentul de stiva
                     ; de la adresa efectivă specificată în BX
mov ss:[di], cx      ;memorează conţinutul lui CX in segmental de stiva la
                     ;adresa de offset dată de conţinutul lui DI
```

**Exemplu:**

```
MOV AX, 1234H
MOV BX, 5678H
MOV CX, 0ABCDH
PUSH AX
PUSH BX
PUSH CX
MOV BX, SP
MOV WORD PTR SS:[BX], 1111H
MOV WORD PTR SS :[BX+1], 2222H
POP CX; CX←2211H
POP BX; BX←5622H
POP AX; AX← 1234H
```

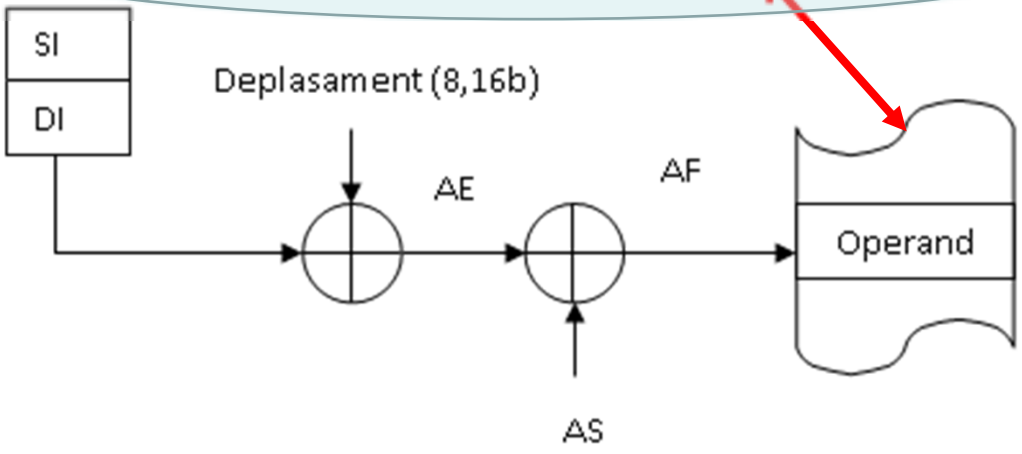


# Adresarea bazată/ Adresarea indexată



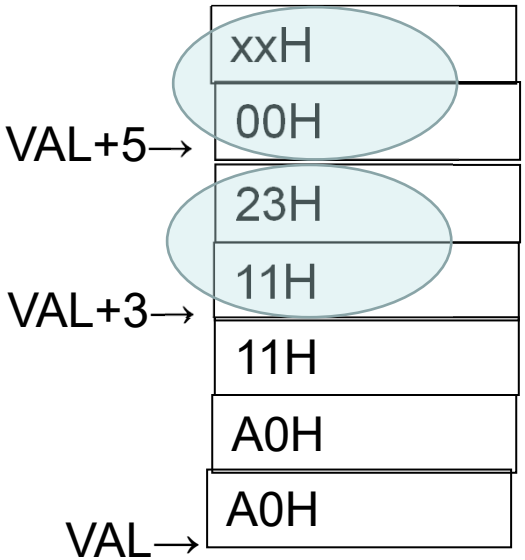

a) Adresarea bazată

Registrul de segment implicit: DS pt. BX, SI, DI
SS pt. BP




b) Adresarea indexată

```
.data
    VAL dw 0A0A0H,1111H, 23h

.code

    ..................

mov   bx,    3
mov   ax,    VAL[bx]

mov   ax,    bx[VAL]
mov   ax,    [bx+VAL]
mov   ax,    [bx].VAL
mov   ax,    [bx+VAL+2]

mov   bx,   offset VAL
mov   ax,    5[bx]
mov   ax,    bx[5]
mov   ax,    [bx+5]
mov   ax,    [bx].5
```



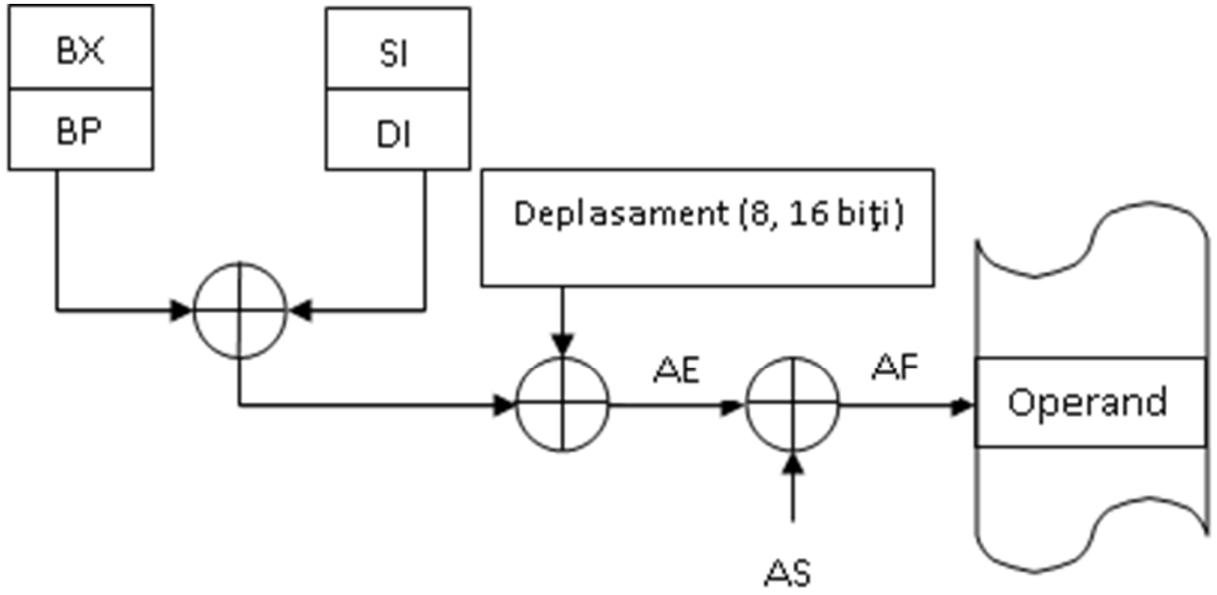

```
.data
    VAL dw 0A0A0H,1111H, 23h

.code

        ..................

    mov   bx,    3
    mov   ax,    VAL[bx]

    mov   ax,    bx[VAL]
    mov   ax,    [bx+VAL]
    mov   ax,    [bx].VAL
    mov   ax,    [bx+VAL+2]

    mov   bx,   offset VAL
    mov   ax,    5[bx]
    mov   ax,    bx[5]
    mov   ax,    [bx+5]
    mov   ax,    [bx].5
```

# Adresarea bazată/ Adresarea indexată

## Exemplu (de accesare a stivei folosind adresarea bazată):

```
mov     bp,   sp
mov     ax,   1234h
push    ax
mov     bx,   5678h
push    bx
mov     byte ptr [bp-2], 0FFH
mov     word ptr [bp-3], 5FA3h
pop     bx      ; bx←0A378h
pop     ax      ; ax←125Fh
```

# Adresarea BAZATĂ ŞI INDEXATĂ




**Registrul de segment implicit: DS pentru SI, DI, BX**
**Registrul de segment implicit: SS pentru BP**

**Exemple:**

```
mov   ax,   [bx][si]
mov   ax,   [bx+si+7]
mov   ax,   [bx+si].7
mov   ax,   [bx][si][7]
mov  byte ptr [bx+di+4], 14h
```

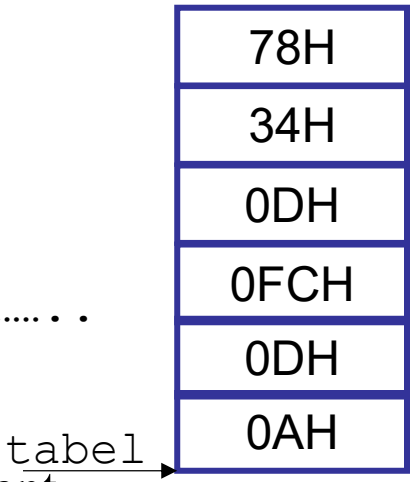
# Rostul modurilor de adresare

Pentru scrierea codurilor relocabile dinamic

Pentru structurarea programelor

# Rostul modurilor de adresare-structurarea programelor

```
.model small
.stack 100
.data
        tabel  DB 0CAh, 13, 0FCh, 13, 52, 78h
.code
start:
        mov    ax, @data
        mov    ds, ax
……..
        mov     cx, 6              ; contorul buclei
        mov     bx, offset tabel
        mov     si, 0              ; indexul primului element
bucla:
        mov     al,[bx+si]   ;mov al, tabel[si]
        ……                   ;prelucreaza data din AL
        inc     si
        loop    bucla
        ……..                 ;se continua programul
```



## Tipuri de date utilizate în limbaj de asamblare

**BYTE** – 1 octet –

Poate fi conţinut: – în memoria internă;

– într-un registru de 8 biţi al procesorului.

***Interpretări:*** – întreg pe 8 biţi cu sau fără semn;

– caracter ASCII.

***Directiva de definire:*** **DB** (Define Byte).

**Exemplu:**

```
Val1 DB 12, ‘A’, ‘ABCDEF’, 10 DUP(‘a’)
……..
MOV AL, VAL1
MOV CX, WORD PTR VAL1+8
MOV DX, WORD PTR VAL1+2;
```

# Tipuri de date utilizate în limbaj de asamblare

**WORD** – 2 octeti

Poate fi conţinut: – în memoria internă;

–într-un registru de 16 biţi al procesorului

***Interpretări:*** – **întreg** pe 16 biţi cu sau fără semn;

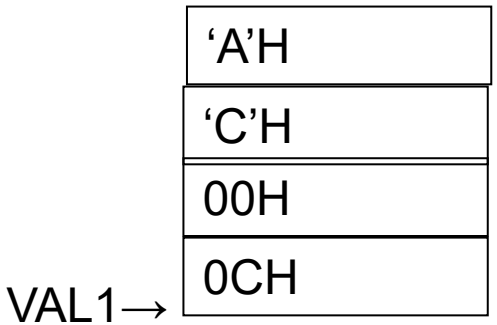
– secvenţă de **două caractere ASCII**;

– **adresă de memorie** de 16 biţi.

***Directiva de definire:*** **DW** (Define Word).

**Exemplu:**

```
Val1 DW 12, 'AC'
……..……
MOV AX, VAL1
```



**DOUBLE-WORD** – 4 octeţi –

Poate fi conţinut: – în memoria internă;
– pereche de registre de 16 biţi;
– registru 32 biţi (procesoarele de 32 biţi).

***Interpretări:*** – întreg pe 32 biţi cu sau fără semn;
– număr real în simplă precizie;
– adresă de memorie de 32 biţi.

***Directiva de definire:*** **DD** (Define Double-Word).

**QUAD-WORD** – 8 octeţi –

Poate fi conţinut: – în memoria internă;

– în pereche de registre de 32 biţi

(procesoare de 32 biţi).

***Interpretări:*** – întreg pe 64 biţi cu sau fără semn;

– număr real în dublă precizie.

***Directiva de definire:*** **DQ** (Define Quad-Word).

## Tipuri de date utilizate în limbaj de asamblare

**TEN-BYTES** – 10 octeţi –

**Poate fi conţinut:** – în memoria internă;

– în registrele coprocesoarelor matematice 80x87.

***Interpretări:*** – număr întreg reprezentat ca secvenţă de cifre **BCD (împachetate)** cu semn memorat explicit;

– număr real în precizie extinsă.

***Directiva de definire:*** **DT** (Define Ten-Bytes).

5433d =1539(16)=0001 0101 0011 1001

5493h=0101 0100 0011 0011 **BCD impachetat**

05040303h=00000101 00000100 00000011 00000011 **BCD despachetat**