Organizarea regiștrilor și a memoriei

Curs #5

📅 Cuprins

- Regiștri generali
  - Data
  - Pointer, index
  - Control
  - Segment
- Adresarea memoriei
- Moduri de lucru procesor
- Întreruperi

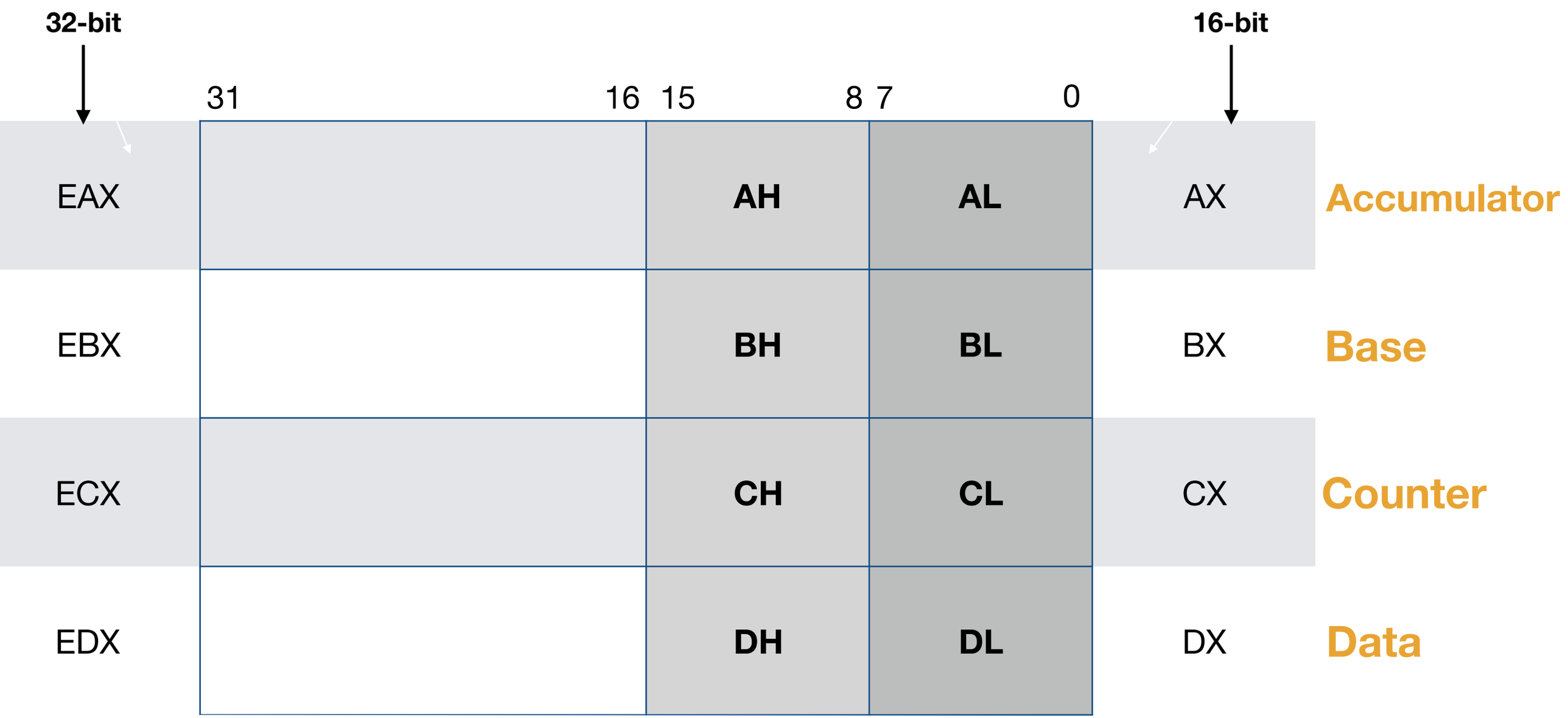
REGIȘTRI GENERALI

- Regiştrii cu scop general participă la operaţii aritmetice sau logice
- În ei se stochează operanzii şi rezultatele operaţiilor.
  - Pe 32 biți: **EAX, EBX, ECX, EDX**
  - Pe 16 biți: **AX, BX, CX, DX**
  - Pe 8 biți: **AH, AL, BH, BL, CH, CL, DH, DL.**

REGIȘTRI GENERALI

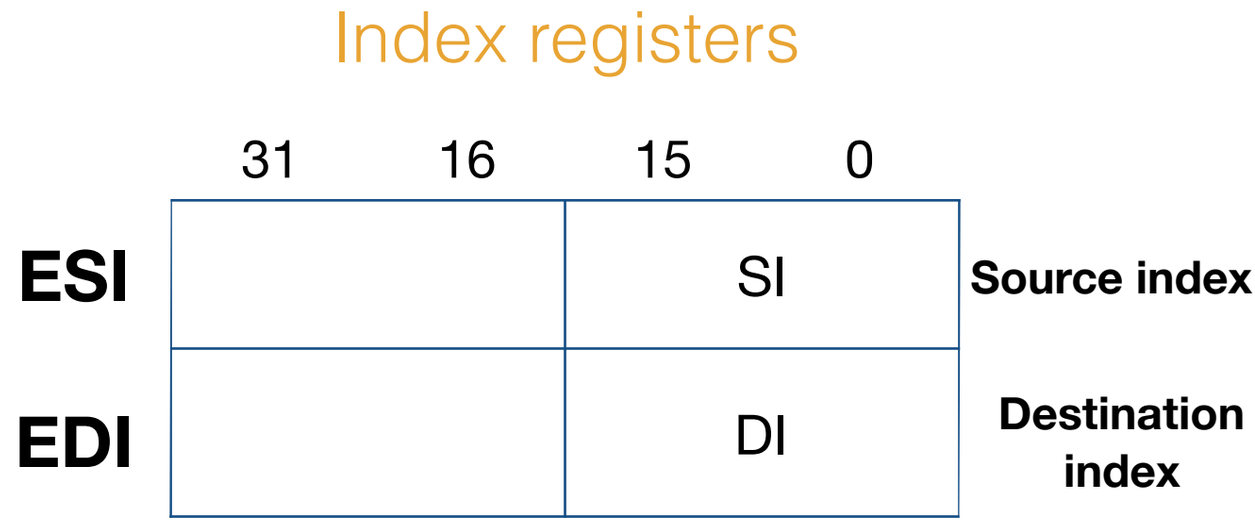
- Unii regiştri generali sunt folosiţi implicit în unele operaţii.
- Astfel:
  - **registrul AX** este registrul acumulator (folosit pentru a păstra rezultatul operaţiilor aritmetice sau logice şi pentru operaţii de transfer date la/de la porturile de I/E).
  - **registrul BX** este folosit ca registru de bază în modul de adresare a memoriei
  - **registrul CX** este folosit ca contor
  - **registrul DX** este folosit în operaţiile de înmulţire şi împărţire şi pentru adresarea porturilor de I/E.

REGIȘTRI GENERALI

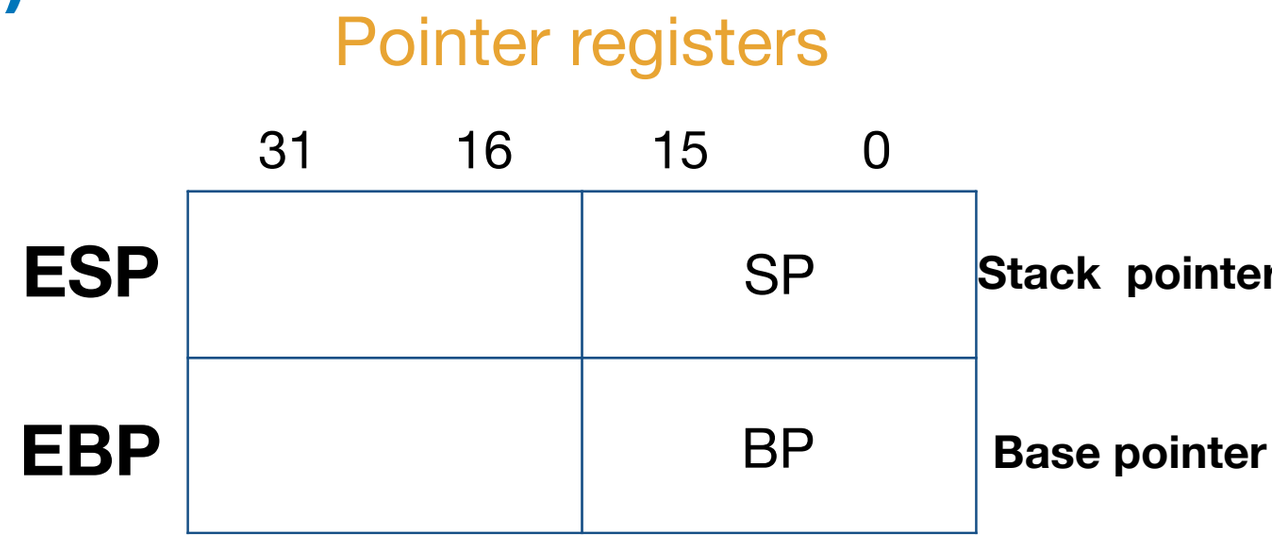


REGIȘTRI GENERALI

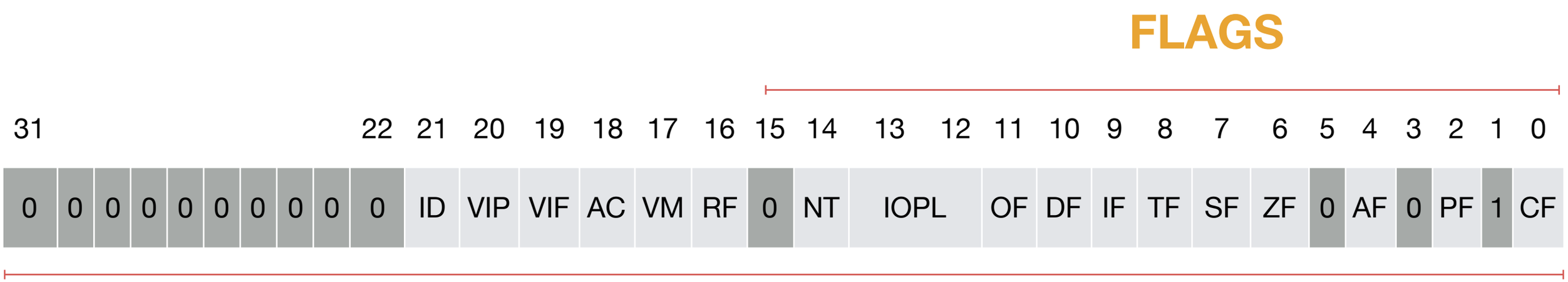
- Două registre index
  - 16 sau 32-biți
  - Instrucțiuni pe stringuri
  - **source (SI); destination (DI)**



- Două registre pointer
  - 16 sau 32-biți
  - Exclusiv pentru stivă



REGIȘTRI DE STARE



STATUS	CONTROL	SYSTEM
CF = Carry Flag	DF = Direction flag	TF = Trap flag
PF = Parity Flag		IF = Interrupt flag
AF = Auxiliary carry flag		IOPL = I/O privilege level
ZF = Zero flag		NT = Nested task
SF = Sign flag		RF = Resume flag
OF = Overflow flag		AC = Alignment check

EFLAGS (Indicatorii de stare)

- **CF (Carry Flag)** - indicator de transport - reflecta transportul in exterior al bitului cel mai semnificativ al rezultatului operatiilor aritmetice. Astfel, acest indicator poate fi folosit in cazul operatiilor in dubla precizie. Valoarea CF = 1 semnifica fie transport la adunare fie imprumut la scadere. De asemenea, indicatorul CF este modificat si de instructiunile de deplasare si rotatie.

- **PF (Parity Flag)** - indicator de paritate - este 1 daca rezultatul are paritate para (contine un numar par de biti 1). Acest indicator este folosit de instructiunile de aritmetica zecimala.

- **AF (Auxiliary Carry Flag)** - indicator de transport auxiliar - este 1 daca a fost transport de la jumatatea de octet inferioara la jumatatea de octate superioara (de la bitul 3 la bitul 4). Acest indicator este folosit de instructiunile de aritmetica zecimala.

EFLAGS (Indicatorii de stare)

- **ZF (Zero Flag)** - indicatorul de zero - este 1 daca rezultatul operatiei a fost zero.

- **SF (Sign Flag)** - indicatorul de semn - este 1 daca cel mai semnificativ bit al rezultatului (MSb) este 1, adica in reprezentarea numerelor in complement fata de 2 (C2) rezultatul este negativ (are semn -).

- **OF (Overflow Flag)** - indicatorul de depasire aritmetica (a gamei de valori posibil de reprezentat) - este 1 daca dimensiunea rezultatului depaseste capacitatea locatiei de destinatie si a fost pierdut un bit (indica la valorile cu semn faptul ca se "altereaza" semnul).

EFLAGS (Indicatorii de control)

- **DF (Direction Flag)** – este utilizat de instrucţiunile pe şiruri şi specifică direcţia de parcurgere a acestora:
  - 0 – şirurile se parcurg de la adrese mici spre adrese mari;
  - 1 – şirurile sunt parcurse invers.

- **IF (Interrupt Flag)** – acest indicator controlează acceptarea semnalelor de întrerupere externă. Dacă IF = 1 este activat sistemul de întreruperi, adică sunt acceptate semnale de întrerupere externă (mascabile, pe linia INTR); altfel, acestea sunt ignorate. Indicatorul nu are influenţă asupra semnalului de întrerupere nemascabilă – NMI.

- **TF (Trace Flag)** – este utilizat pentru controlul execuţiei instrucţiunilor în regim pas cu pas (instrucţiune cu instrucţiune), în scopul depanării programelor. Dacă indicatorul este 1, după execuţia fiecărei instrucţiuni se va genera un semnal de întrerupere intern (pe nivelul 1). Evident, execuţia secvenţei de tratare a acestei întreruperi se va face cu indicatorul TF = 0.

Registre de control

- **EIP**
  - Instruction pointer (instrucțiunea curentă)
- **EFLAGS**
  - Status flags
    - Se actualizează după operații aritmetice/logice
  - Direction flag
    - Forward/backward direcția copierii
  - System flags
    - IF: activare întreruperi
    - TF: Trap flag (pentru debugging)

Registre segment

- 16 biți
- Memoria segmentată
- Conținut distinct
  - *Code*
  - *Data*
  - *Stack*

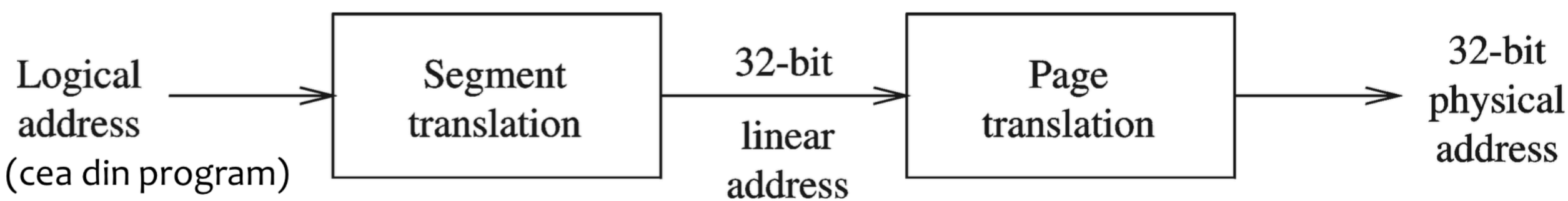
15	0
CS	*Code segment*
DS	*Data segment*
SS	*Stack segment*
ES	*Extra segment*
FS	*Extra segment*
GS	*Extra segment*

Modurile x86

- Toate procesoarele Intel actuale au două moduri importante de lucru:
  - **Modul “real”**
    » Adrese și registre pe 16 biţi
    » Memorie 1MB
    » Folosit după reset (grub)
  - **Modul protejat**
    » Registre de 32 biţi
    » Segmentare și paginare
    » Protecţia kernelului și a proceselor
    » Folosit de Linux, Windows

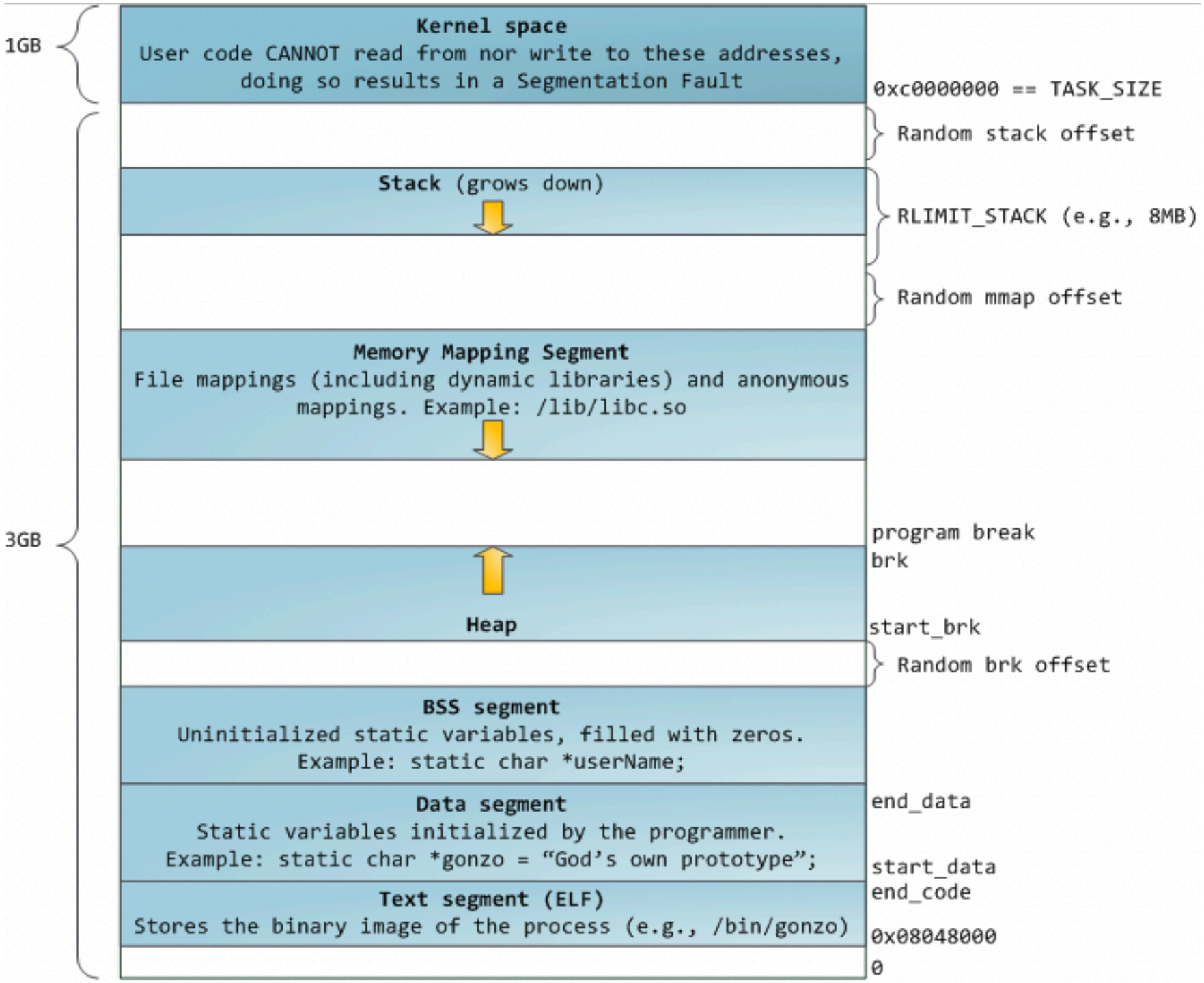
Modul protejat

- Segmentarea & paginarea = translatare adrese 32 biţi
- Segmentarea: adrese logice → adrese lineare
- Paginarea: adrese lineare → adrese fizice
- Segmentare, paginare -> curs SO

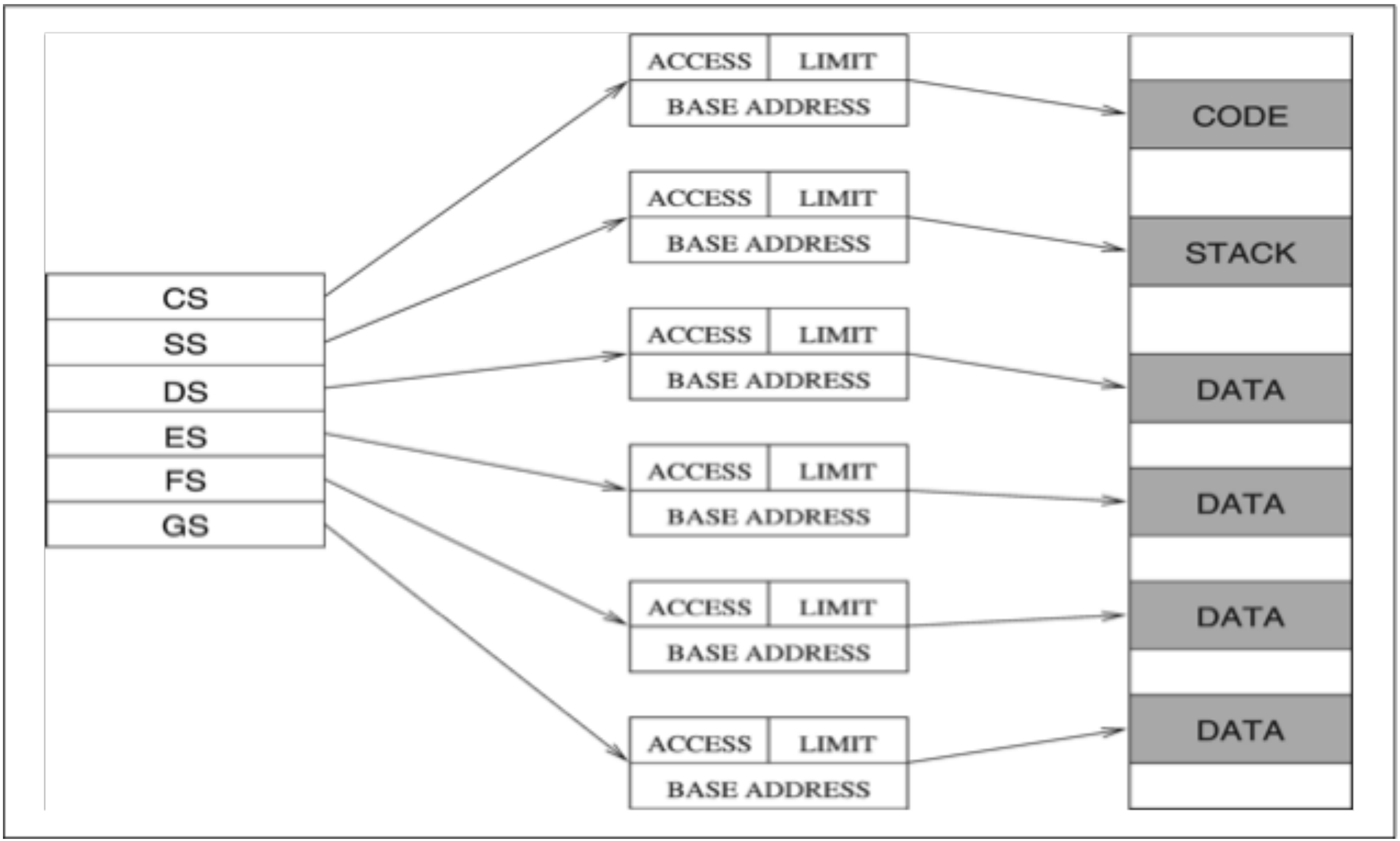


Imaginea unui proces in memorie

Adrese logice
(în program)

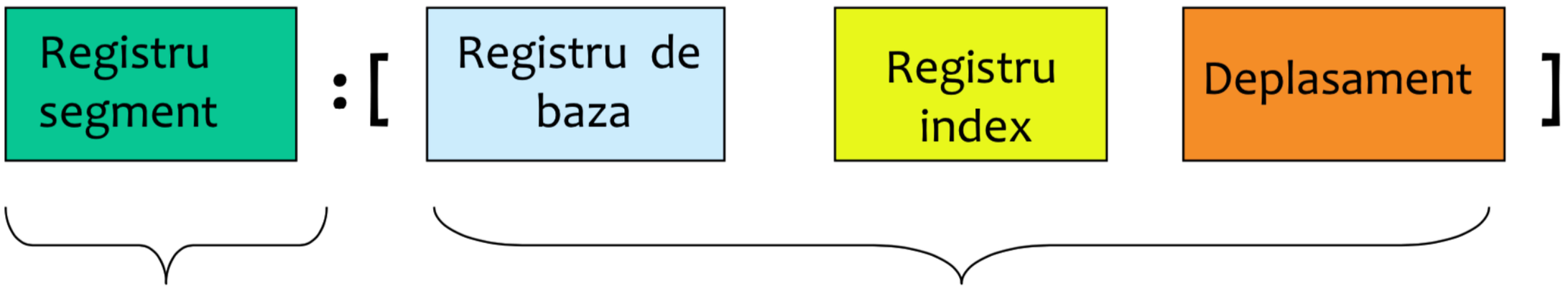


Adresarea memoriei



Adresarea memoriei

calculul adresei logice



Adresarea memoriei

$$\begin{Bmatrix} CS: \\ DS: \\ SS: \\ ES: \\ FS: \\ GS: \end{Bmatrix} \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{Bmatrix} \right] + \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} * \begin{Bmatrix} 1 \\ 2 \\ 4 \\ 8 \end{Bmatrix} \right] + [\text{displacement}]$$

```
    eax                 ebx   esi
; un dword
    ebx
; adresa lui mybuffer, deci valoare imediată
```

Exemplu - adresare directă

```
section .data
        var: DD 34
section .text
        mov eax, var ; put var's >>address<< into the eax register
        add eax, var ; add to eax, the >>address<< of var
```

```
int var = 34;
eax = &var; /* mov eax, var */
eax = eax + &var; /* add eax, var */
```

Exemplu - adresare indirectă

```
section .data
        var: DD 34
section .text
        mov eax, [var] ; put var's >>value<< into eax
        add eax, [var] ; add to eax, the >>value<< of var
```

Acest lucru ar fi echivalent în **C** cu:

```
int var = 34;
eax = var; /* mov eax, [var] */
eax = eax + var; /* add eax, [var] */
```

Procesarea întreruperilor

Procesarea întreruperilor

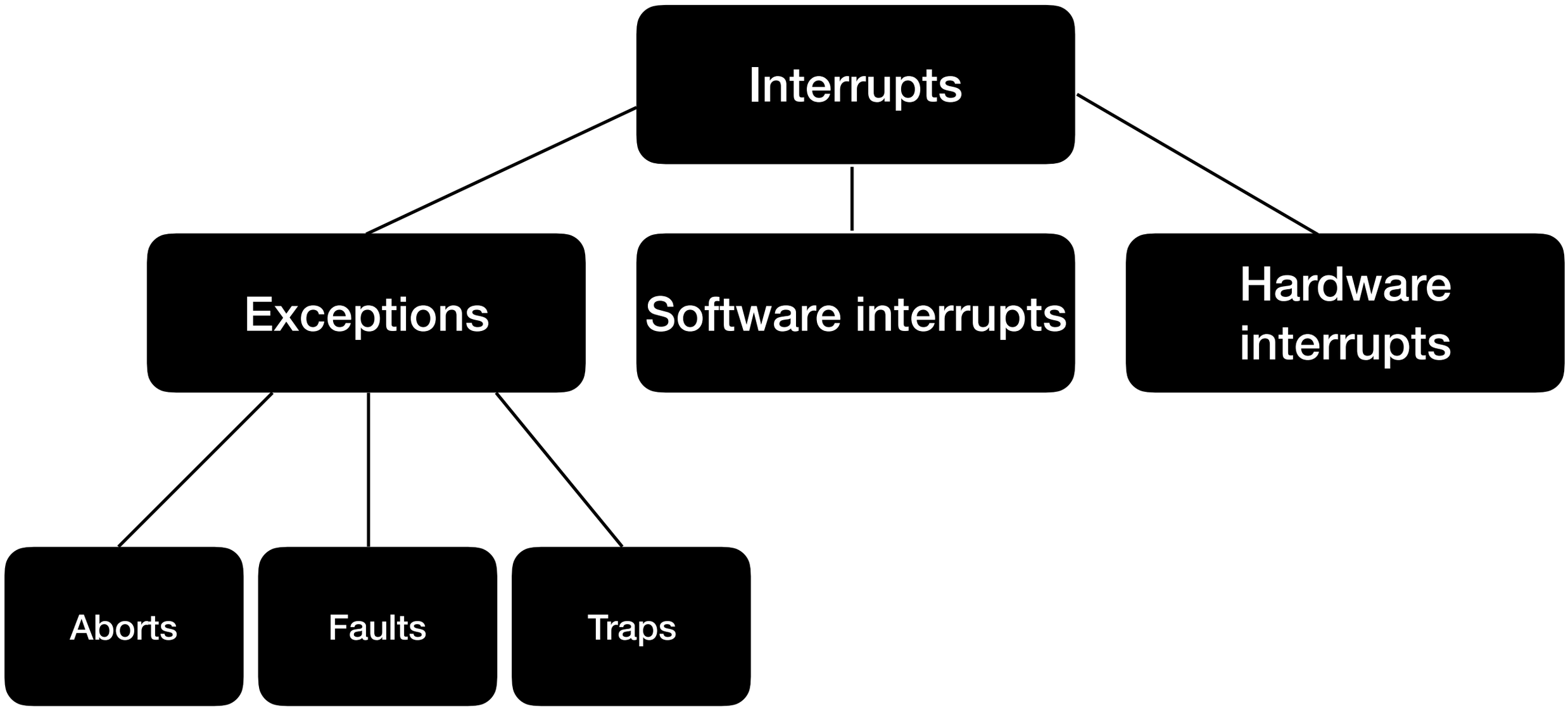
- Întreruperile alterează fluxul programului
  - * Comportament similar apelului de procedură
- Întreruperea transferă controlul către ISR
  - * ISR = *interrupt service routine*, *interrupt handler*
- La terminarea ISR programul original continuă
- **Întreruperile** = mod eficient de a trata evenimente neanticipate

Întreruperi vs. Proceduri

- Inițiate de *software* sau *hardware*
- Tratează evenimente *anticipate* și *neanticipate*
- ISR sunt mereu în memorie
- Sunt identificate cu numere
- Registrul EFLAGS salvat automat

- Inițiate doar de *software*
- Tratează evenimente *anticipate* din program
- Sunt încărcate cu programul
- Sunt identificate cu nume
- Nu salvează registrul EFLAGS

Taxonomia întreruperilor



Declanșarea unei întreruperi

1. Push EFLAGS onto the stack
2. Clear IF & TF (interrupt and trap flags)
   * Se dezactivează alte întreruperi
3. Push CS and EIP onto the stack
4. se încarcă CS:EIP din IDT (*interrupt description table*)

Interrupt Flag

- **IF** (Interrupt flag) [dez]activează întreruperile
- IF==0 → nu se permit întreruperi
  - Instrucţiunea `cli` (*clear interrupts*)
  - La declanșarea intreruperii IF devine 0
  - Instrucţiunea `sti` (*set interrupts*)

Întoarcerea din ISR

- Ultima instrucţiune din ISR este `iret`
- Acţiunile executate la `iret` sunt:
  1. Pop EIP
  2. Pop CS
  3. Pop EFLAGS
- ISR sunt responsible pentru
  * A restaura TOATE registrele folosite
  * A nu lăsa date pe stivă

Excepții

- 3 tipuri: **Fault**, **Trap**, **Abort**
- Fault și trap sunt declanșate între instrucţiuni

- Abort e declanșată la erori severe
  - Erori hardware
  - Valori inconsistente în sistem

Fault și Trap

- **Fault**
  - » Se declanșează înainte de instrucţiunea în cauză
  - » Se repornește instrucţiunea
  - » Exemplu: *page-not-found fault*

- **Trap**
  - » Se declanșează după instrucţiunea în cauză
  - » Nu se repornește instrucţiunea
  - » Exemplu: *Overflow exception*
  - » Exemplu: *întreruperi definite de utilizator*

Numere de întreruperi rezervate

Întrerupere	Scop
0	Divide error
1	Single-step
2	Nonmaskable interrupt (MNI)
5	Breakpoint
6	Overflow
13	General protection exception
14	Page fault
16	Floating point error

Întreruperi rezervate

- **Divide Error Interrupt**
  * Instrucţiunea div/idiv câtul nu încape în destinaţie
- **Single-Step Interrupt**
  * Dacă Trap Flag este setat (TF==1)
    - CPU generează automat int 1 după execuţia fiecărei instrucţiuni
  * folositor pentru debugger
- **Breakpoint Interrupt**
  * int 3 în cod mașină este un octet(CCH)

- **Page fault Interrupt**
  * Se accesează o pagină virtuală care nu este încă mapată
  * SO aduce pagina sau termină procesul

Întreruperi software

- Iniţiate de execuţia instrucţiunii:

  `int interrupt-number`

  - interrupt-number = [0 .. 255]

- în Linux `int 0x80` este apelul de sistem

  * 180 servicii diferite
  * EAX conţine numărul serviciului

- Funcţional similare cu apelurile de proceduri