
Arhitectura x86



De citit: Dandamudi
capitolele 3, 14

Modificat: Oct-14-18

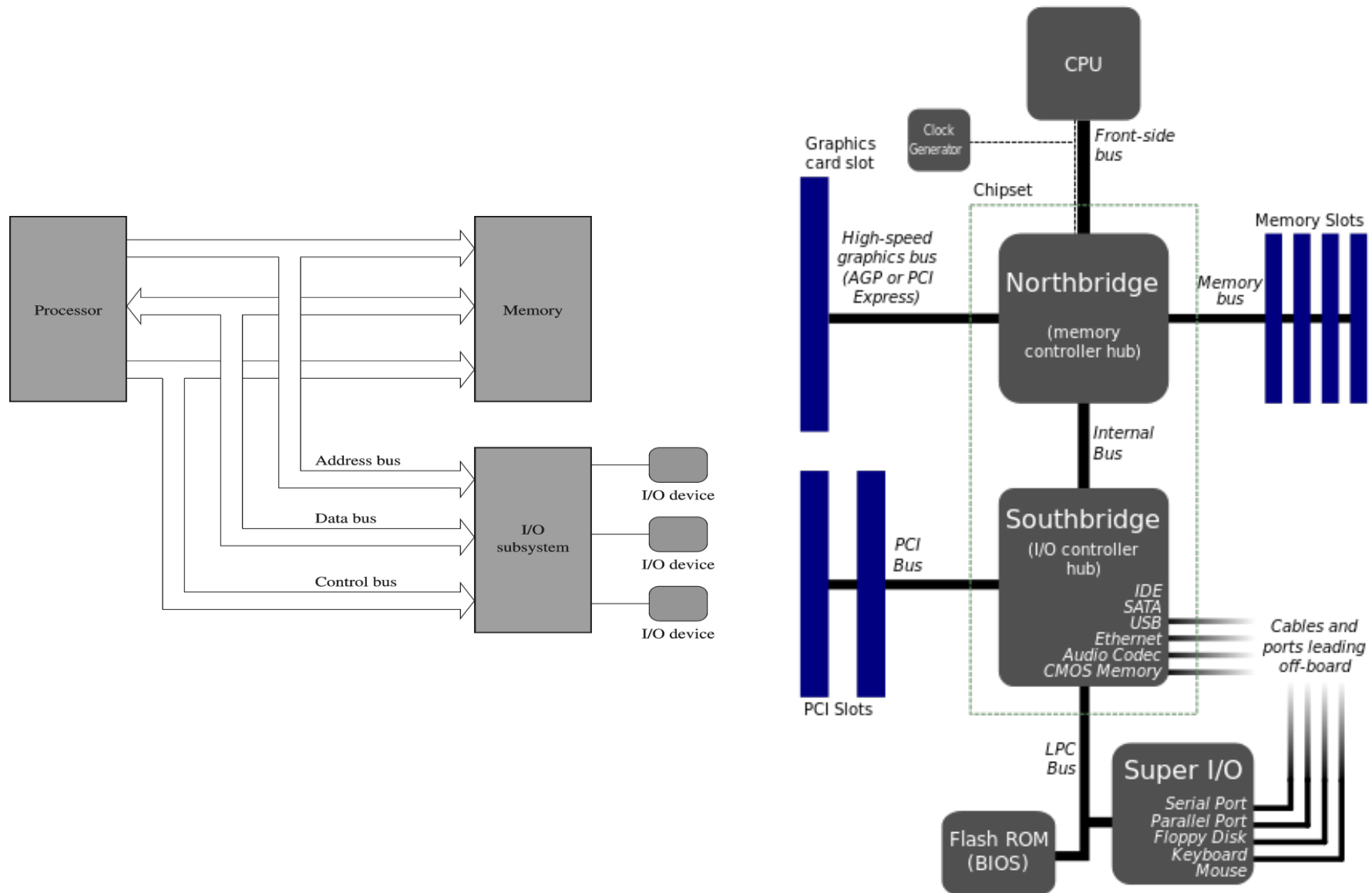
Cuprins curs 2

- Familia x86
- Registrele
 - * Data
 - * Pointer, index
 - * Control
 - * Segment
- Modul protejat
 - * Registrele Segment
 - * Descriptori de segment
 - * Tabele de descriptori
 - * Modele de segmentare
- Modul real
- Segmentare, Paginare
- Întreruperi
- Demo sasm, gdb
- Instrucțiuni mov, add, jmp

Istoricul procesoarelor Intel

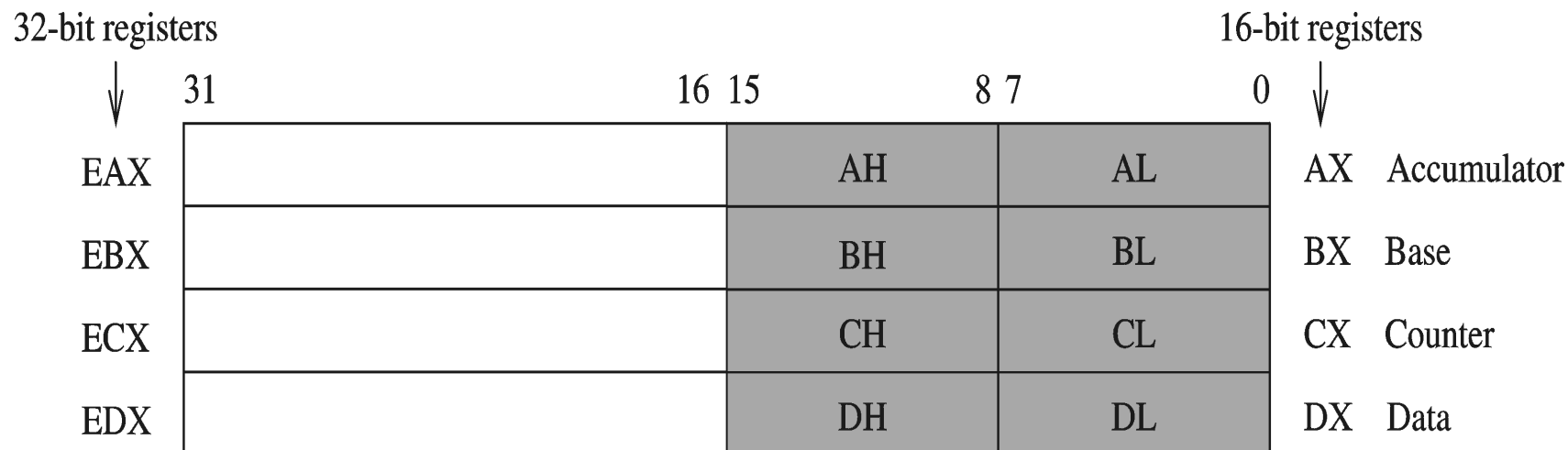
Procesor	An	Frecvența	Tranzistoare	Registre	Bus date	Max addr
4004	1969	0.74	2.3K	4	12	4K
8080	1974	2	4.5K	8	16	64K
8086	1978	8	29K	16	16	1 MB
80386	1985	20	275K	32	32	4 GB
Pentium	1993	60	3.1M	32	40?	4GB
Pentium 4	2000	1500	42M	32		64GB
Core 2	2006	3000	291M	64		64GB
Core i7	2008	3400	1.4G	64		64GB
Xeon E5	2012	3600	5.5G	64		768GB

Arhitectura x86



x86 registre pe 32 biți

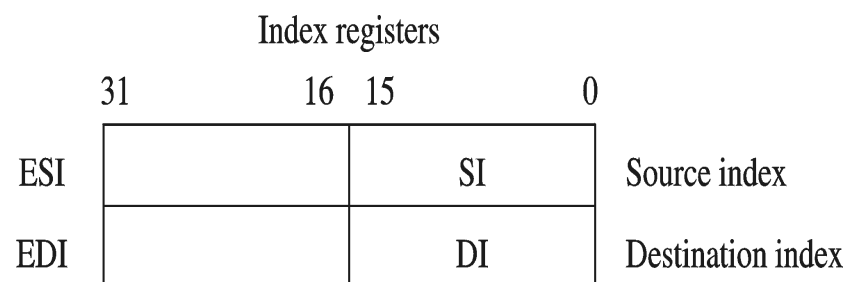
- Registre de 32 biți pot fi folosite
 - * Pe 32 biți (EAX, EBX, ECX, EDX)
 - * Pe 16 biți (AX, BX, CX, DX)
 - * Pe 8 biți (AH, AL, BH, BL, CH, CL, DH, DL)
- Unele registre au utilizări speciale
 - * ECX este numărător pentru instrucțiunea loop



x86 registre pe 32 biți

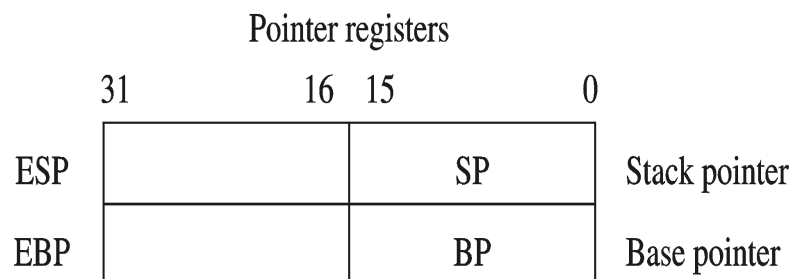
- Două registre index

- * 16 sau 32 biți
- * Instrucțiuni pe stringuri
- * source (SI); destination (DI)
- * Pot fi folosite în scop general



- Două registre pointer

- * 16 sau 32-biți
- * Exclusiv pentru stivă



1. *Journal of the American Medical Association*, 1997; 277: 1001-1005.

(c) \mathcal{C}_1 is a \mathcal{C}_2 -subalgebra of \mathcal{C}_1 .

1. *Journal of the American Medical Association*, 1997; 278: 1039-1044.

Table 1

Journal of Management Education 36(8) 907-924

[illegible]

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.
- **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 instrucțiunile pe șiruri și specifică direcția de parcurgere a acestora:
 - * 0 – șirurile se parcurg de la adrese mici spre adrese mari;
 - * 1 – șirurile 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 influență asupra semnalului de întrerupere nemascabilă – NMI.
- **TF** (Trace Flag) – este utilizat pentru controlul execuției instrucțiunilor în regim pas cu pas (instrucțiune cu instrucțiune), în scopul depanării programelor. Dacă indicatorul este 1, după execuția fiecărei instrucțiuni se va genera un semnal de întrerupere intern (pe nivelul 1). Evident, execuția secvenței de tratare a acestei întreruperi se va face cu indicatorul TF = 0.

x86 registre pe 32 biți

- 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 intreruperi

- TF : Trap flag (pentru debugging)

x86 registre pe 32 biți

- 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 86

- **Toate** procesoarele x86 au două moduri importante

- * Modul “real”

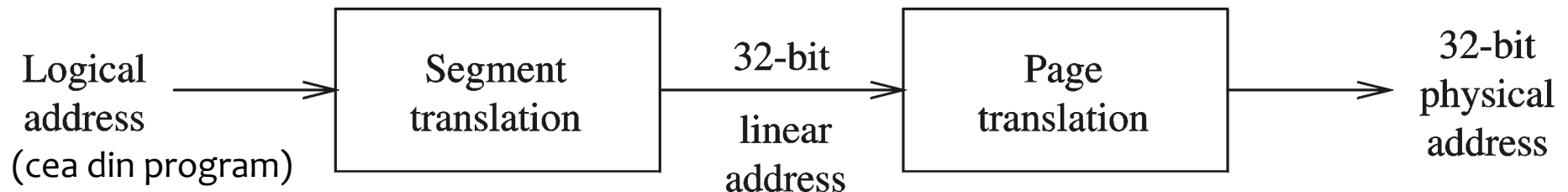
- » Adrese și registre pe 16 biți
- » Memorie 1MB
 - 640K should be enough for everyone
- » Folosit după reset (grub)

- * Modul protejat

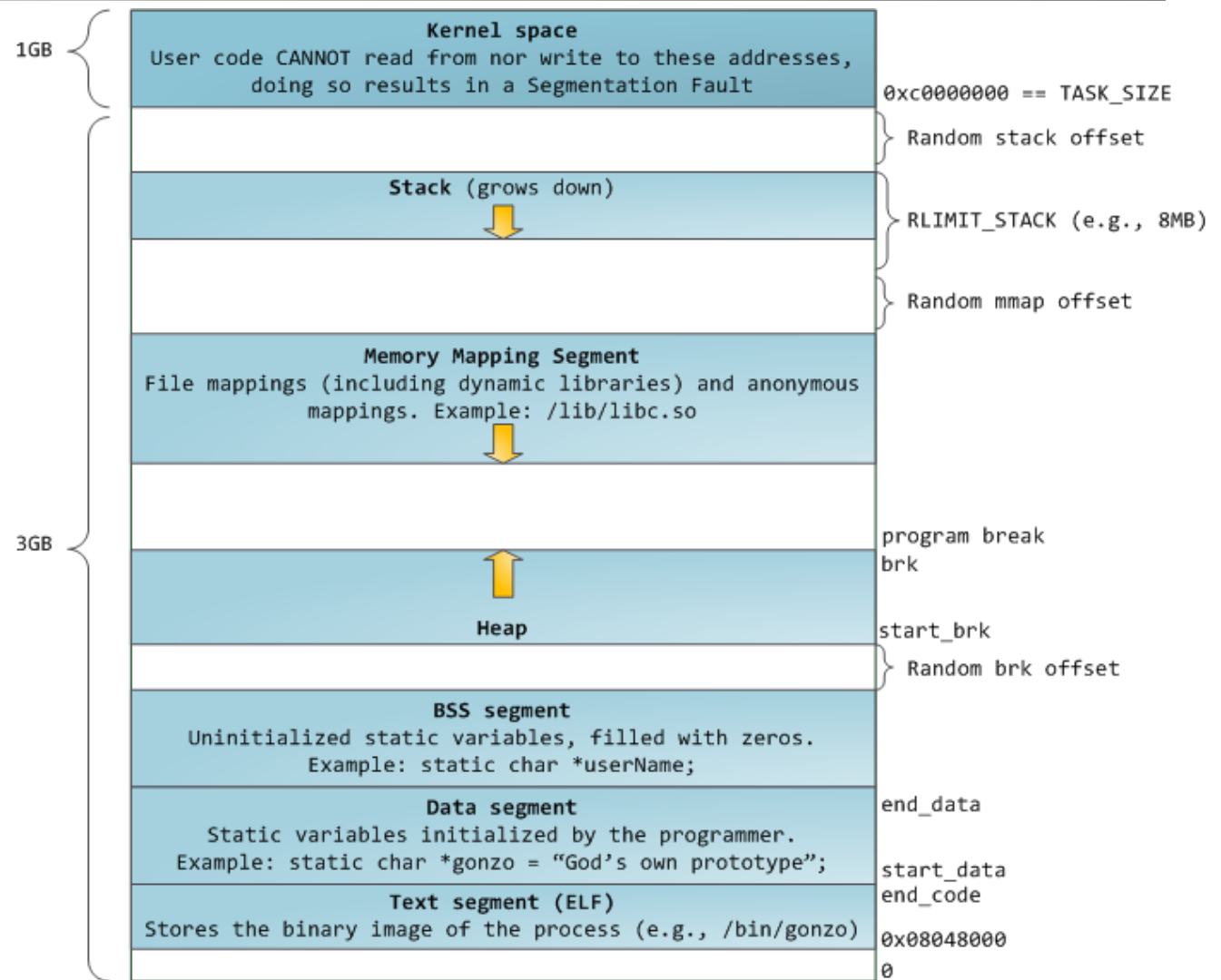
- » Registre de 32 biți
- » Segmentare și paginare
- » **Protecția** kernelului și a proceselor
- » Folosit de Linux, Windows

Modul Protejat

- Segmentarea&paginarea = **translatare** adrese 32 biți
- Segmentarea: adrese logice → adrese lineare
- Paginarea: adrese lineare → adrese fizice
- Segmentare, paginare -> cursurile SO, SO2

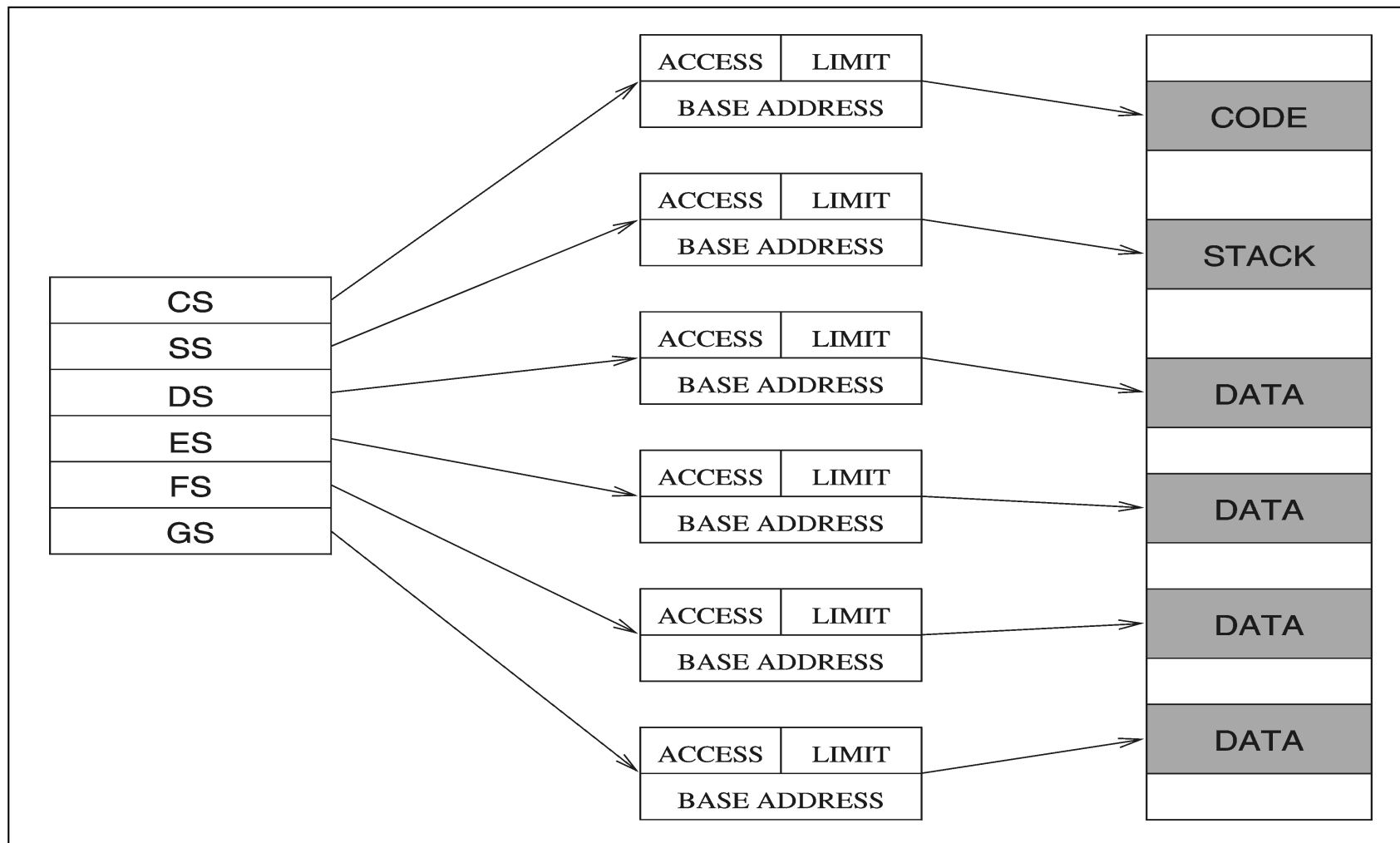


Imaginea unui proces în memorie



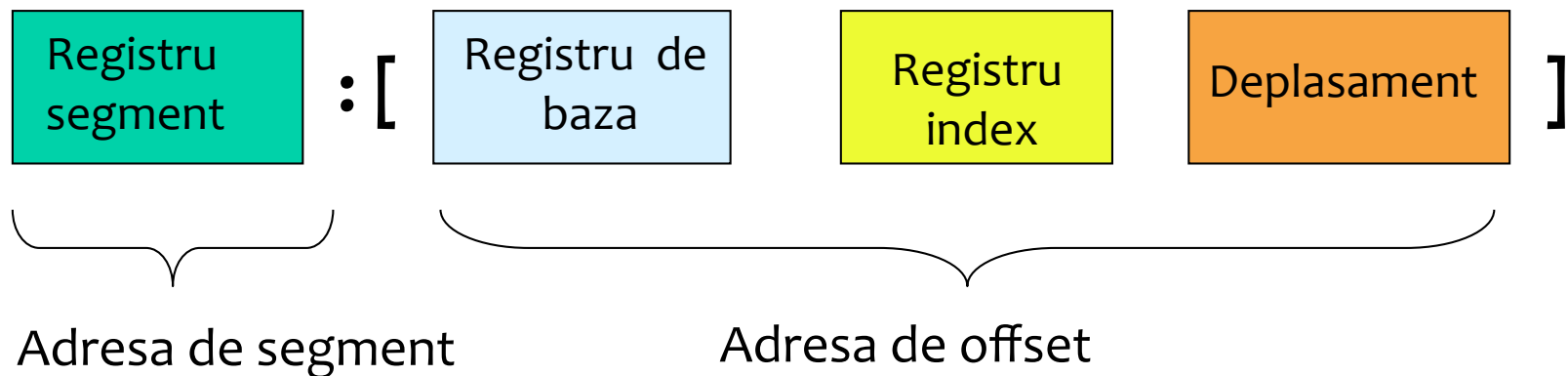
Adrese logice
(în program)

Adresarea memoriei



Adresarea memoriei

calculul adresei logice



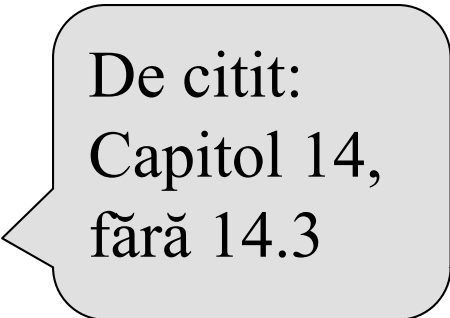
Adresarea memoriei

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

```
mov eax, [mybuffer + ebx + esi*4 + 9]  
; un dword
```

```
mov ebx, mybuffer  
; adresa lui mybuffer, deci valoare imediată
```

Procesarea întreruperilor



De citit:
Capitol 14,
fără 14.3

Ce sunt întreruperile?

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

Intreruperi vs. Proceduri

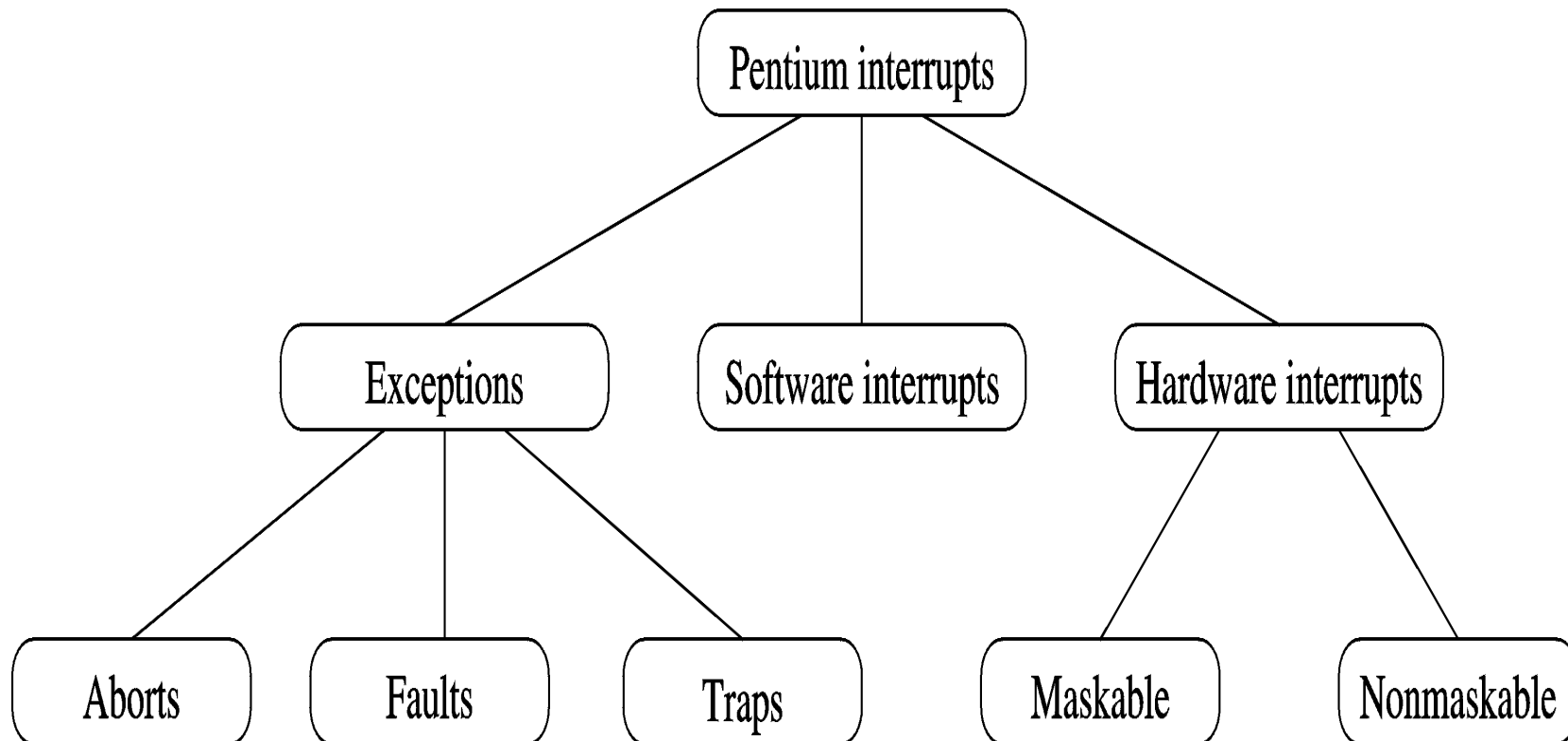
Intreruperi

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

Proceduri

- 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
 - * Instrucțiunea **c*l*i** (clear interrupts)
 - * La declanșarea intreruperii IF devine 0
 - * Instrucțiunea **s*t*i** (set interrupts)

Întoarcerea din ISR

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

Excepții

- 3 tipuri: Fault, Trap, Abort
- Fault și trap sunt declanșate **între** instrucțiuni
- Abort e declanșată la erori severe
 - * Erori hardware
 - * Valori inconsistente în sistem

Fault și Trap

- Fault

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

- Trap

- » Se declanșează **după** instrucțiunea în cauză
- » **NU se repornește** instrucțiunea
- » Exemplu: Overflow exception (întreruperea 4)
- » 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
 - * Instrucțiunea div/idiv câtu nu încap în destinație
- Single-Step Interrupt
 - * Dacă Trap Flag este setat (TF==1)
 - » CPU generează automat int 1 după execuția fiecărei instrucțiuni
 - * folositor pentru debugger
- Breakpoint Interrupt
 - * `int 3` în cod mașină este un octet(CCH)
 - * Cum este folosită la debug?
- Page fault Interrupt
 - * Se accesează o pagină virtuală care nu este încă mapată
 - * SO aduce pagina sau termină procesul

Întreruperi software

- Inițiate de execuția instrucțiunii

int interrupt-number

interrupt-number = [0 .. 255]

- în Linux **int 0x80** este apelul de sistem
 - * 180 servicii diferite
 - * EAX conține numărul serviciului
- Funcțional similare cu apelurile de proceduri

Exemple `int 0x80` (Linux)

- * Tastatura și ecranul sunt tratate ca fișiere
- * Fișiere standard, deja deschise
 - » Standard input (**`stdin`**), descriptor 0
 - Dispozitiv asociat: tastatura
 - » Standard output (**`stdout`**) descriptor 1
 - Dispozitiv asociat: terminal
 - » Standard error (**`stderr`**) descriptor 2
 - Dispozitiv asociat: terminal

Exemple `int 0x80` (Linux)

- Citește din fișier

System call 3

Inputs: EAX = 3

EBX = file descriptor

ECX = pointer to input buffer

EDX = max. # of bytes to read

Returns: EAX = # of bytes read

Error: EAX = error code

Exemple `int 0x80` (Linux)

- Scrie în fișier

System call 4

Inputs: EAX = 4

EBX = file descriptor

ECX = pointer to output buffer

EDX = # of bytes to write

Returns: EAX = # of bytes written

Error: EAX = error code

Procedura putch

; primește caracterul in AL.

putch:

pusha

mov [temp_char],AL

mov EAX,4 ; 4 = write

mov EBX,1 ; 1 = std output (display)

mov ECX,temp_char ; pointer to char buffer

mov EDX,1 ; # bytes = 1

int 0x80

popa

ret

Procedura getstr

; primește EDI = buffer, ESI = lungimea

getstr:

pusha

pushf

mov EAX,3 ; file read service

mov EBX,0 ; 0 = std input (keyboard)

mov ECX,EDI ; pointer to input buffer

mov EDX,ESI ; input buffer size

int 0x80

dec EAX

mov byte[EDI+EAX],0 ; append NULL character

popf

popa

ret

Întreruperi hardware

- Sunt asincrone, produse de hardware
 - * Se aplică un semnal extern procesului
- Întreruperile hardware pot fi
 - * Maskable
 - * Non-maskable
 - » Întreruperea rezervată 2 (**NMI**)

Cum se declanșează întreruperile hardware?

- Întreruperile Non-maskable sunt declanșate de pinul **NMI** al procesorului
 - * Procesorul răspunde întotdeauna
 - * Nu poate fi dezactivat prin program
- Întreruperile Maskable sunt declanșate de pinul **INTR** al procesorului
 - * declanșate doar dacă $IF == 1$
 - * Mascate cu cli, activate cu sti

Demo

- Programul bc – conversii folosind obase, ibase
- `курс-02-demo` – `sasm`
 - * Registrele EAX, AX, AH, AL
 - * EFLAGS
 - * EIP
 - * Instrucțiunile MOV, ADD, JMP
- `курс-02-demo` – Makefile + gdb
 - * Comenzile `b main`, `r`, `n`
 - * `set $eax = 0xffffffff`
 - * `set $eip = main`
 - * Decomentați instrucțiunea `jmp` și reasamblați

Demo

- Programul bc – conversii folosind obase, ibase
- `curs-02-demo` – `sasm`
- `curs-02-demo` – `Makefile` + `gdb`
- Atenție la `~/.gdbinit`
 - * `sasm` folosește `gdb`!
 - * `sasm` doar introductiv

Intrebări?

