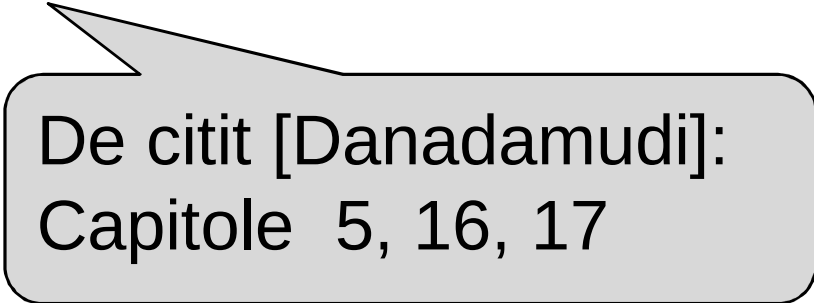

Stiva, Funcții, C + assembler



De citit [Danadamudi]:
Capitole 5, 16, 17

Modificat: 15-Oct-18

Proceduri și utilizarea stivei

Chapter 5
S. Dandamudi

Cuprins

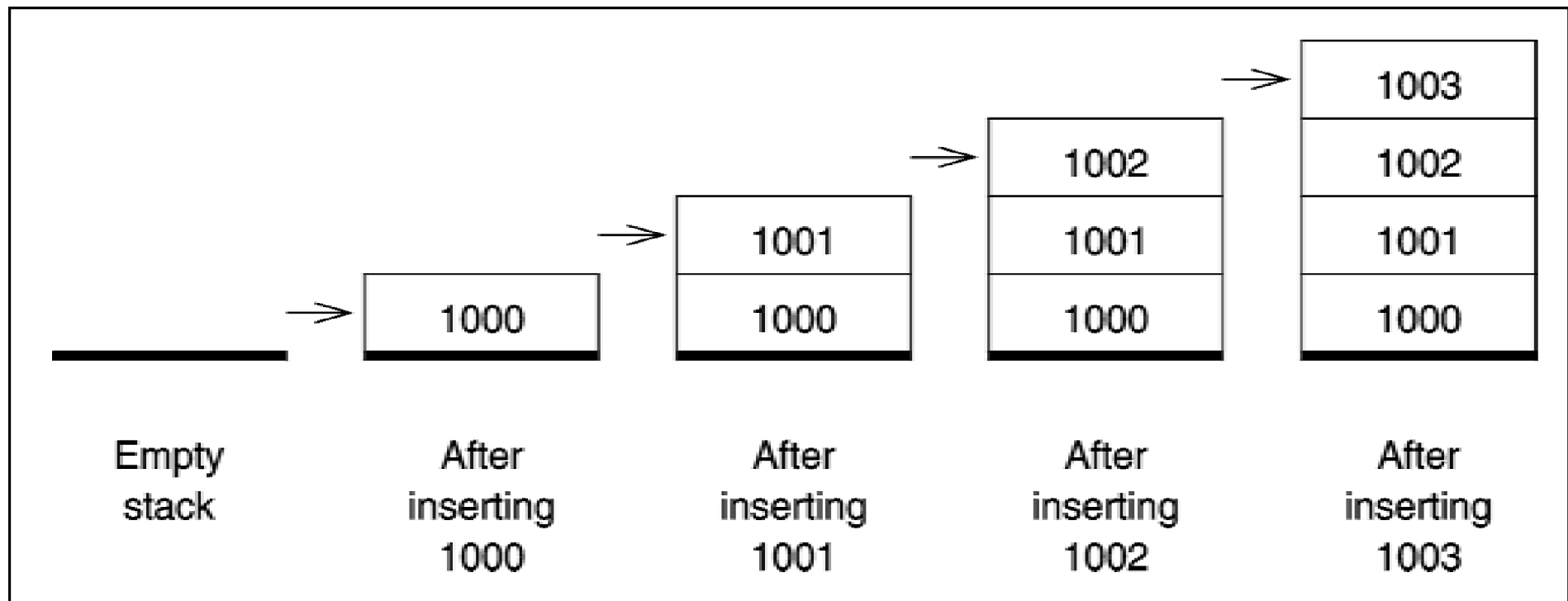
- Ce este stiva?
- Implementarea stivei pentru x86(Pentium)
- Instrucțiuni de lucru
- Utilizările stivei
- Proceduri
 - * Instrucțiuni x86
- Transmiterea parametrilor prin:
 - * registre
 - * stiva
- Exemple
 - * Apelul prin valoare
 - * Apelul prin referință
 - * Sortare prin metoda bulelor
- Proceduri cu număr variabil de parametri
- Variabile locale
- Program cu multiple module sursă
- Performanța: Overhead-ul procedurilor

Ce este stiva?

- Stiva este o coadă last-in-first-out (LIFO)
- Dacă vizualizăm stiva ca un vector de elemente, atunci inserția și ștergerea sunt restricționate la unul din capetele vectorului
- Numai elementul din vârful stivei (en. top-of-stack a.k.a TOS) este direct accesibil
- Structura implementează două operații de baza:
 - * push (inserție)
 - * pop (ștergere)

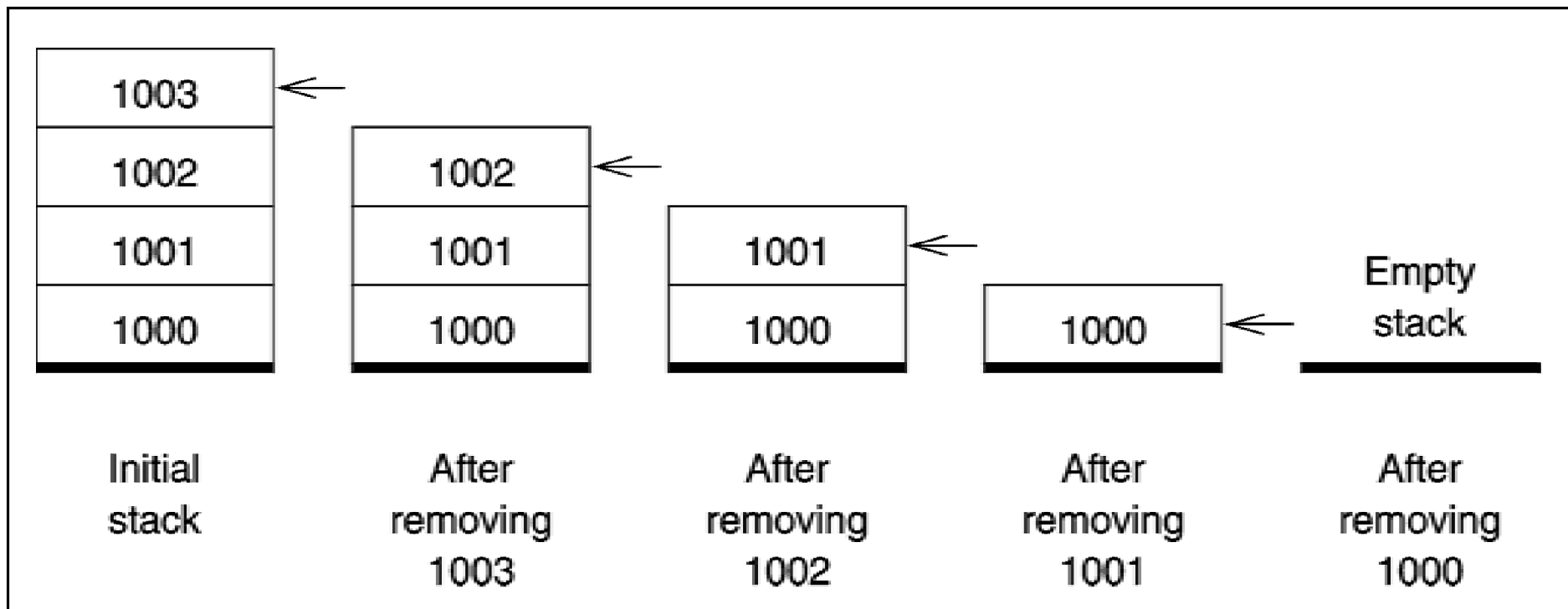
Ce este stiva? (cont'd)

- Exemplu
 - * Inserarea elementelor in stivă
 - » Săgeata pointează către vârful stivei



Ce este stiva? (cont'd)

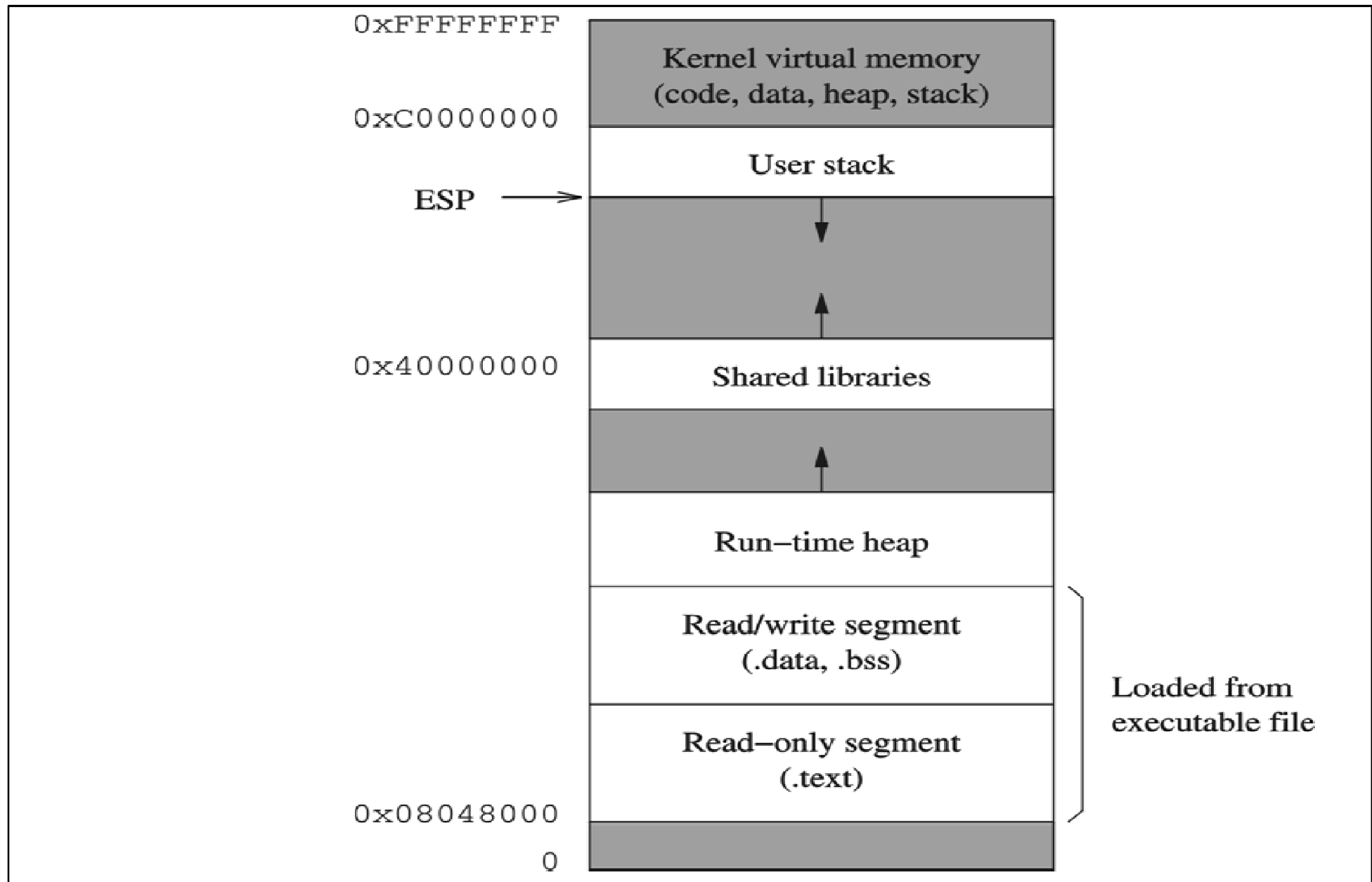
- Exemplu
 - * Ștergerea elementelor din stivă
 - » Săgeata pointează către vârful stivei



Implementarea stivei pentru x86

- La x86 este utilizat un registru segment rezervat:
 - * Registrul SS (Stack Segment) indică adresa de start a segmentului, iar registrul ESP (Extended Stack Segment) indica deplasamentul față de adresa start a vârfului stivei
 - * Impreuna SS:ESP indică vârful stivei
- Caracteristicile implementării pentru x86:
 1. Date de tip word/16-bit sau doubleword/32-bit
 2. Stiva crește **spre adrese mai mici “in jos”**
 3. TOS pointează către ultimul element introdus in stivă

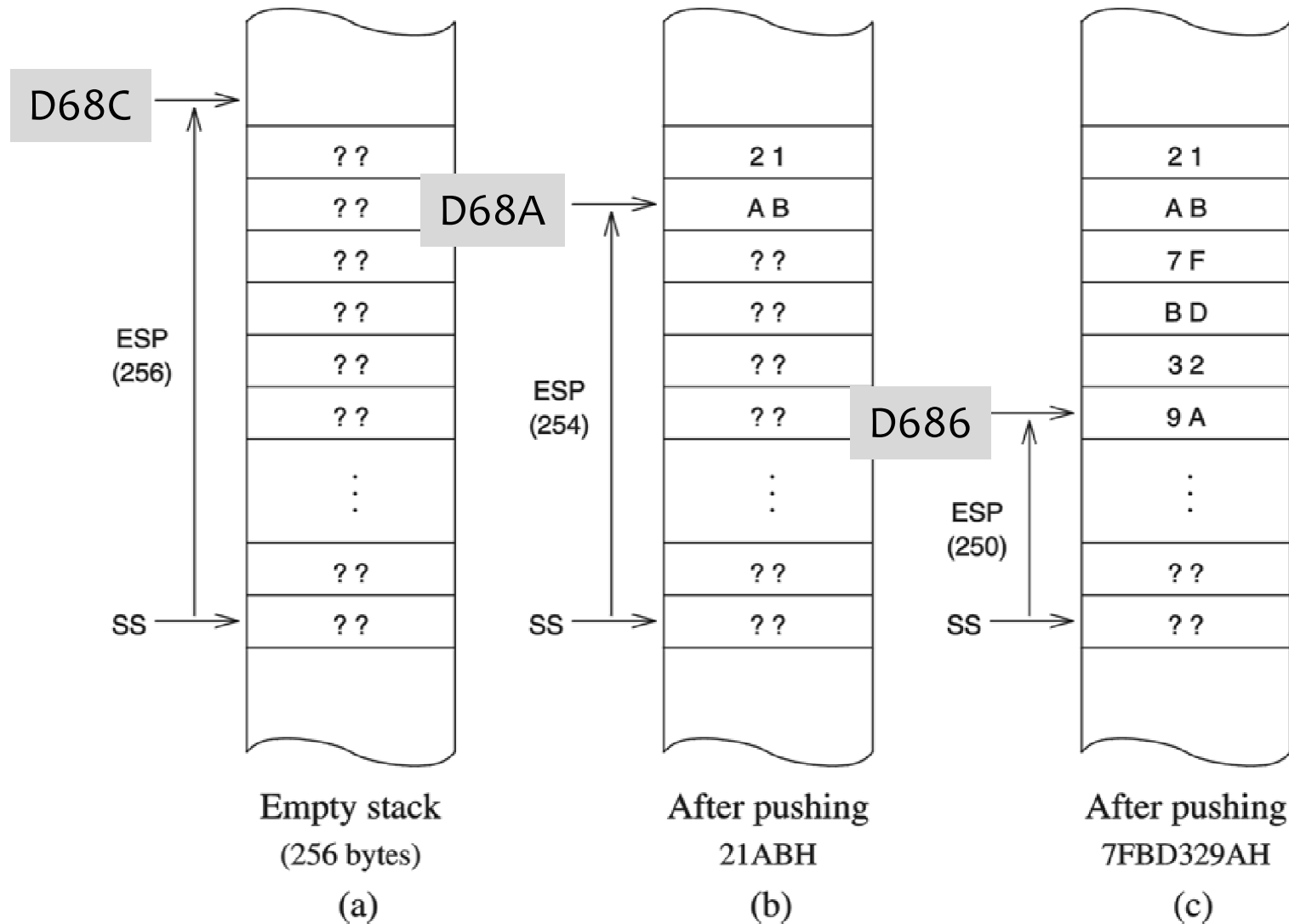
Harta memoriei pentru un proces in Linux



Instrucțiuni de lucru cu stiva

- x86 permite două instrucțiuni de **bază**:
 push sursa
 pop destinație
- **Sursa și destinația** pot fi
 - * registre de uz general 16- sau 32-bit (ex: ax sau eax)
 - * registru de segment (ex: DS, SS, CS, etc...)
 - * un word sau double word din memorie (ex: word [var+2])
- **source** în plus poate fi și o data imediată pentru instrucțiunea **push** de lungime 8, 16, sau 32 bit
 - *push 0xAB

Exemplu de lucru cu stiva pentru x86 - 1



Instructiuni de lucru cu stiva: Example

- Pentru o stivă goală următoarea secvență de instrucțiuni **push**

push word 21ABH

push 7FBD329AH

rezultă în starea prezentată la (c) în figura anterioară

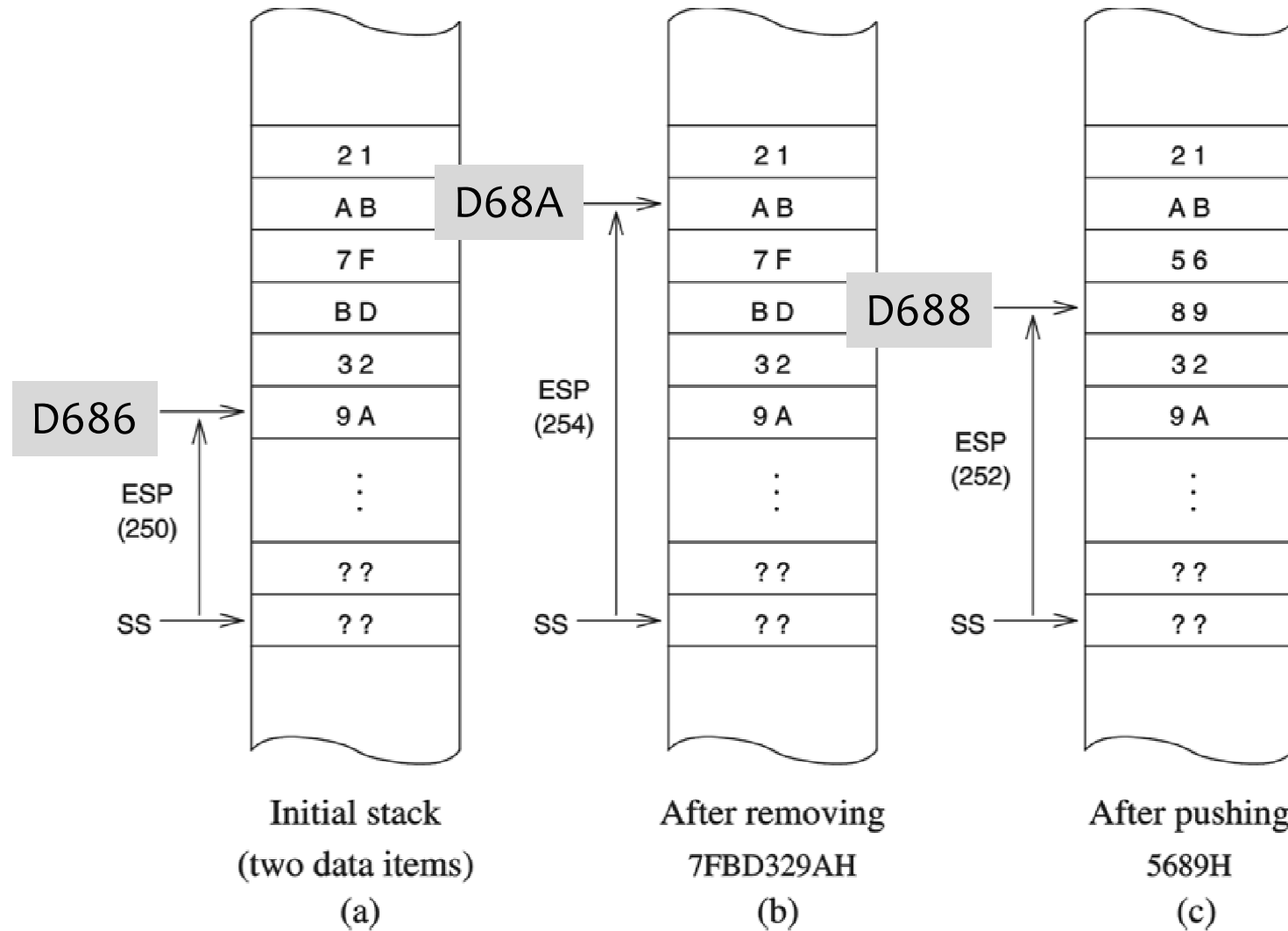
- După execuția instrucțiunii:

pop EBX

Rezultă în starea stivei de la (b) din figura următoare, iar EBX va conține valoarea 7FBD329AH

- Demo SASM: test_stack_s.asm

Exemplu de lucru cu stiva pentru x86 - 2



Instrucțiuni adiționale

Operații de stiva asupra fanioanelor

- Instrucțiunile **push** si **pop** nu pot fi utilizate cu registrul de stare (EFLAGS)
- Doua instructiuni speciale in acest sens sunt:
pushfd (push 32-bit flags)
popfd (pop 32-bit flags)
- Operanzii nu sunt necesari
- Se folosesc **pushfw** and **popfw** pentru 16-bit (FLAGS)

Instructiuni aditionale

Operatii de stiva asupra la toti registrii de uz general

- Instrucțiunile **pushad** si **popad** sunt folosite pentru a salva si a restaura 8 registre de uz general
EAX, ECX, EDX, EBX, ESP, EBP, ESI, and EDI
- **pushad** executa o operație de push pentru fiecare din registrii anteriori în ordinea data (EAX primul și EDI ultimul)
- **popad** restaureaza toti registrii exceptand registrul ESP
- Se folosesc **pushaw** si **popaw** pentru a executa aceeași operație pentru registre la nivel de 16-bit (AX primul și DI ultimul)

Utilizările stivei

- Trei utilizari de baza:
 1. Stocarea datelor temporare
 2. Transferarea controlului
 3. Transmiterea parametrilor

1. Stocarea datelor temporare

Exemplu: Inter-schimbarea variabilelor **value1** si **value2** poate fi realizata utilizând stiva pentru a salva datele temporare

```
push  value1
push  value2
pop   value1
pop   value2
```

Utilizarea stivei (cont'd)

- Des utilizata pentru eliberarea unor registre

;salveaza EAX & EBX pe stiva

push EAX

push EBX

;EAX si EBX pot fi acum folositi

mov EAX,value1

mov EBX,value2

mov value1,EBX

mov value2,EAX

;restaureaza EAX & EBX din stiva

pop EBX

pop EAX

...

Utilizarile stivei (cont'd)

2. Transferarea Controlului

- Pentru proceduri și întreruperi adresa de retur este salvata pe stiva
- Discuția pentru apelul procedurilor va clarifica în detaliu acesta utilizare

3. Transmiterea Parametrilor

- Stiva este extensiv utilizata pentru transmiterea parametrilor către proceduri
- Discuția de mai târziu va arata cum se realizează acest proces

Proceduri

- Doua tipuri
 - * Apelul-prin-valoare
 - » Primește numai valori
 - » Similar functiilor matematice
 - * Apelul-prin-referință
 - » Primește pointeri
 - » Manipulează direct zona de memorie a parametrilor

Instrucțiuni de lucru cu Procedurile

- x86 dispune de doua instructiuni: **call** si **ret**
- Instrucțiunea **call** este utilizata pentru a apela o procedura, iar formatul acesteia este:

call **proc-name**

nexti: ...

proc-name - numele procedurii (adresa acesteia)

nexti - adresa instructiunii urmatoare

- Acțiunile realizate la apelul unei proceduri:

push **nexti** ; push return address

jmp **proc-name** ; EIP of the procedure

Instrucțiuni de lucru cu Procedurile (cont'd)

- Instrucțiunea **ret** este utilizata pentru a transfera controlul către procedura apelanta
- De unde stie procesorul unde sa se intoarca?
 - * Foloseste adresa de retur salvata pe stiva la executia instructiunii **call**
 - * Este important ca TOS sa arate către acesta adresa în momentul execuției instrucțiunii **ret**
- Actiunile realizate la executia lui **ret** sunt:

add	ESP, 4	; pop return address
jmp	[ESP-4]	; from the stack

Instrucțiuni de lucru cu proceduri

- Putem specifica un întreg optional instrucțiunii **ret**

- * Formatul acesteia este

ret optional_uint

- * Exemplu:

ret 8

- Acțiunile realizate în acest caz sunt :

add	ESP, 4 + optional_uint
jmp	[ESP – 4 - optional_uint]

Cum este transferat controlul in program?

Offset machine code main:



00000000200000007

1D-07 = 16

Diagram illustrating the conversion of a 32-bit hexadecimal value to a 16-bit value. The 32-bit value is 00000000200000007. The 16-bit value is 1D-07 = 16. The diagram shows the 16-bit value being the lower 16 bits of the 32-bit value.

Diagram illustrating the assembly code for the `sum` procedure:

```

sum:
0000001D  55          push EBP
          ret
          ; end of sum procedure
  
```

The code starts with the label `sum:`. The first instruction is `push EBP`, which pushes the base pointer register onto the stack. The second instruction is `ret`, which returns from the procedure. The final line is a comment: `; end of sum procedure`.

2D-1D=10 avg:

00000028 E8F0FFFFFF call sum
0000002D 89D8 mov EAX,EBX

 ; end of avg procedure

Transmiterea parametrilor

- Transmiterea parametrilor este **diferita** fata de limbajele de nivel înalt (C / C++ / Java)
- In limbaj de asamblare
 - » Toți parametrii necesari trebuie dispuși într-o zona de stocare care poate fi accesata mutual de **apelant** și **apelat** (caller vs callee)
 - » Apoi se apeleaza procedura (a.k.a. **call proc_name**)
- Tipuri zone de stocare
 - » Registre (se utilizeaza registrii de uz general)
 - » Memorie (este folosita stiva)
- Doua metode de transmitere a parametrilor:
 - » Metoda prin registre
 - » Metoda care utilizeaza stiva

Transmiterea parametrilor prin registre

- Procedura **apelantă** plasează toți parametrii necesari în registre de uz general înainte de a realiza apelul propriu-zis prin instrucțiunea **call**
- Exemple:
 - * **Demo: PROCEX1.ASM**
 - » apelul-prin-valoare utilizand metoda registrelor
 - » o procedura care realizeaza o suma simpla
 - * **Demo: PROCEX2.ASM**
 - » apelul-prin-referenta folosind metoda registrelor
 - » procedura pentru calculul lungimii unei string

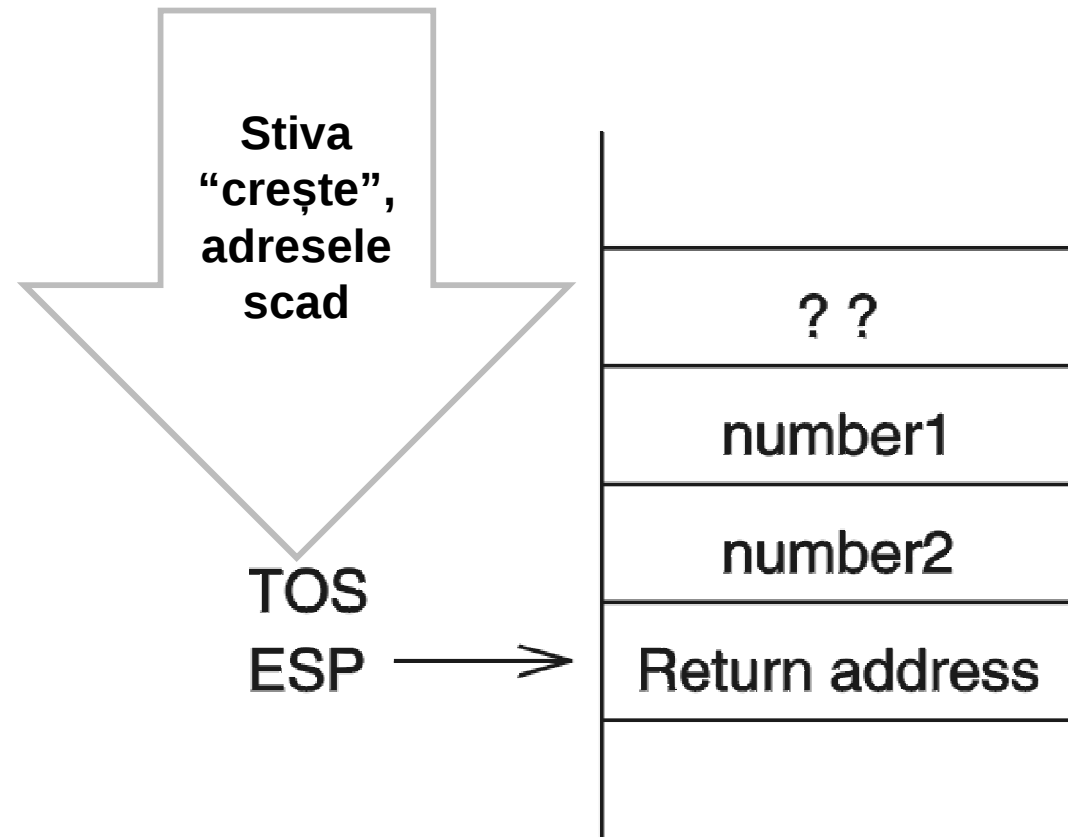
pro și contra pentru metoda registrelor

- Avantaje:
 - * Simplu si convenient
 - * Mai rapid
- Dezavantaje:
 - * Numai un număr limitat de parametri poate fi transferat prin registre
 - Un număr foarte mic de registre este accesibil
 - * Cel mai adesea registrele nu sunt liberi
 - eliberarea acestora prin salvarea lor pe stiva neaga al doilea avantaj al metodei

Transmiterea parametrilor prin stivă

- Toate valorile sunt puse pe stiva înainte de a apel
- Example:

```
push number1  
push number2  
call sum
```



Accesarea parametrilor de pe stivă

- Valorile parametrilor se găsesc pe stiva
- Putem folosi următoarea instrucțiune pentru a accesa valoarea parametrului **number2**

mov EBX, [ESP+4]

Problema: ESP se schimbă cu operațiile **push/pop**

- » Deplasamentul relativ depinde de operațiile efectuate asupra stivei
- » A se evita indexarea după ESP
- Există o alternativă mai bună?
 - * Folosirea lui EBP pentru a accesa parametrii de pe stivă

Folosirea lui EBP pentru acces la parametri

- Abordarea preferata pentru a accesa parametrii:

```
mov  EBP, ESP
```

```
mov  EAX, [EBP+4]
```

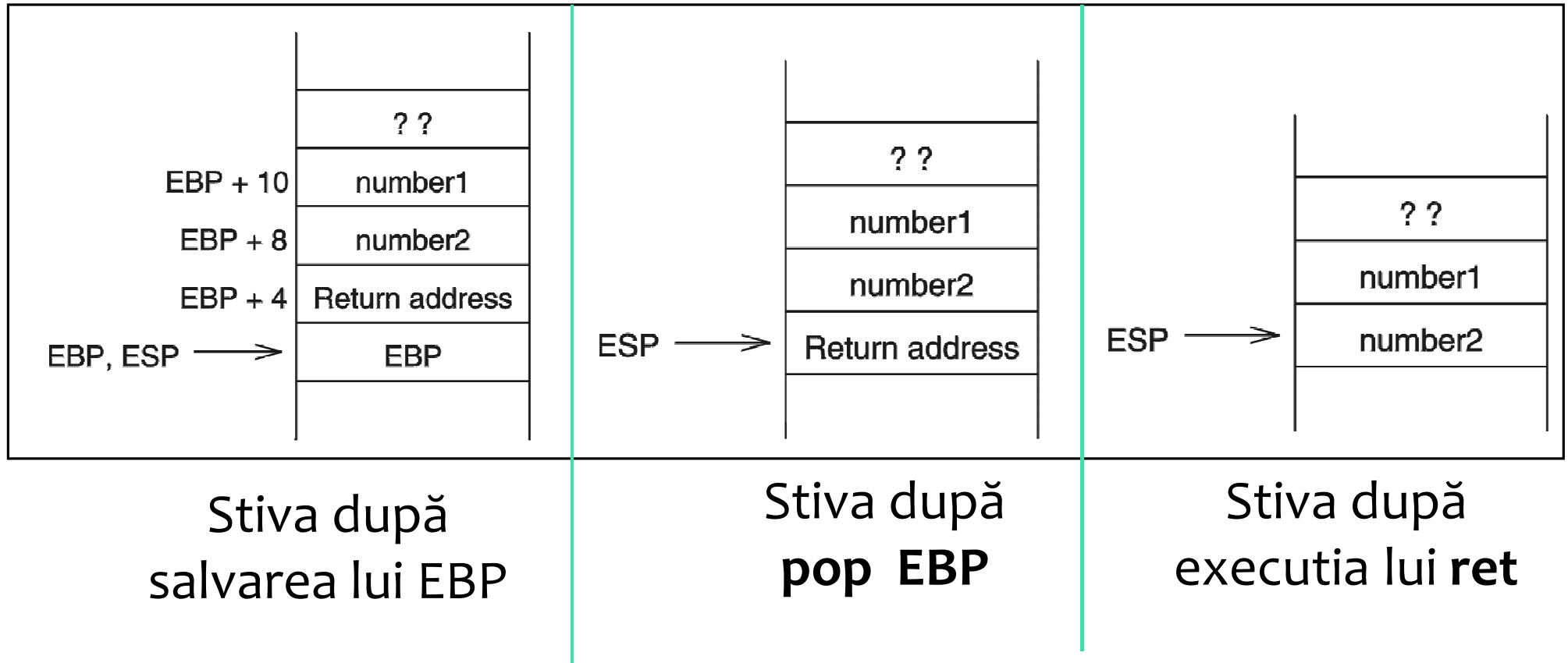
pentru a accesa **number2** din exemplul anterior

- Problema: Continutul lui EBP este pierdut!
 - * Trebuie salvat conținutul lui EBP
 - * Folosim stiva (atenție: se schimbă deplasamentul ESP)

```
push EBP
```

```
mov  EBP, ESP
```

Eliberarea stivei de parametri



Eliberarea stivei de parametri (cont'd)

- Două feluri pentru eliberarea stivei de parametri:
 - * Folosirea intregului optional pentru instrucțiunea **ret**
 - » Folosim
ret 4
pentru exemplul anterior
 - * Adunarea unei constante la ESP în procedura apelantă
(C folosește aceasta metoda)

```
push  number1
push  number2
call  sum
add   ESP,4
```

Probleme de întreținere a stivei

- Cine ar trebui să curețe stiva de parametri?
 - * **Procedura apelantă (caller)**
 - » Trebuie să actualizeze ESP la fiecare apel de procedură
 - » Nu este neapărat necesară dacă procedura are un număr fix de parametri
 - » C utilizează această metodă datorită folosirii unui număr variabil de parametri
 - * **Procedura apelată (callee)**
 - » Codul devine modular (curățarea parametrilor realizată într-un singur loc)
 - » Nu poate fi utilizată cu un număr variabil de parametri

Probleme de întreținere a stivei

- Trebuie salvat conținutul pentru procedura apelantă
 - » Stiva este utilizata în acest scop
- Care dintre registre trebuie salvate?
 - * Se salvează acele registre care sunt utilizate de procedura apelantă și sunt modificați de cea apelată
 - » Atenție: setul de registre utilizat variază în timp
 - * Se salvează totii registrii utilizand **pusha**
 - » Latența crescuta (**pusha** se executa în 5 cicluri de ceas, în timp ce salvarea unui sigur registru se executa într-unul)

Probleme de întreținere a stivei

- Unde se menține starea procedurii apelante?
 - * Procedura apelantă (caller)
 - » Trebuie cunoscute registrele utilizate de procedura apelata
 - » Trebuie incluse instrucțiuni de salvare și restaurare a registrelor la fiecare apel de procedura
 - » **Cauzează probleme de mentenanță a programului**
 - * Procedura apelată (callee)
 - » Metoda preferată deoarece codul devine modular (prezervarea stării se realizează într-un singur loc)
 - » Se evita problemele de mentenanță

Probleme de întreținere a stivei

- Conservarea stării apelantului în timpul apelului
 - » Pe stivă
- Ce registre ar trebui salvate?
 - * Se salvează acele registre care sunt utilizate de apelant (caller) și modificate de apelat (callee)
 - » Poate cauza probleme
 - * Se salveaza toate registrele (metoda brute force)
 - » folosind **pusha**
 - » Latență crescută
 - **pusha** se execută în 5 cicluri de ceas, iar salvarea unui registru doar într-unul sigur

Probleme de întreținere a stivei

Starea stivei după
pusha

??	
number1	EBP + 38
number2	EBP + 36
Return address	EBP + 32
EAX	EBP + 28
ECX	EBP + 24
EDX	EBP + 20
EBX	EBP + 16
ESP	EBP + 12
EBP	EBP + 8
ESI	EBP + 4
EDI	

EBP, ESP →

Instrucțiuni pentru cadrul de stivă

- Instrucțiunea ENTER

- * Facilitează alocarea unui cadru de stivă

enter bytes,level

bytes = spațiu local de stocare

level = nivelul de intercalare (folosim nivelul 0)

- * Exemplu

enter XX,0

este echivalent cu

push EBP

mov EBP,ESP

sub ESP,XX

Instrucțiuni pentru cadrul de stivă

- Instrucțiunea LEAVE
 - * Dealoca un cadru de stiva

leave

- » Nu are operanzi
- » Echivalenta cu

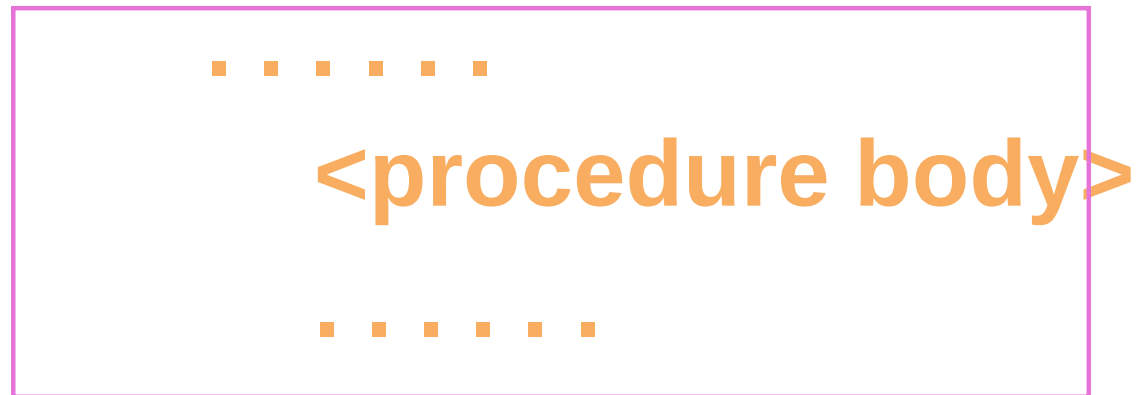
mov ESP,EBP

pop EBP

Schita unei proceduri tipice

proc-name:

enter **XX**,0



<procedure body>

leave

ret **YY**

Transmiterea parametrilor prin stiva - ex.

- **PROCEX3.ASM**

- * apelul prin valoare folosind stiva
- * o procedura pentru calculul sumei

- **PROCSWAP.ASM**

- * apelul prin referința folosind stiva
- * primele doua caractere ale string-ului de input sunt inter-schimate

- **BBLSORT.ASM**

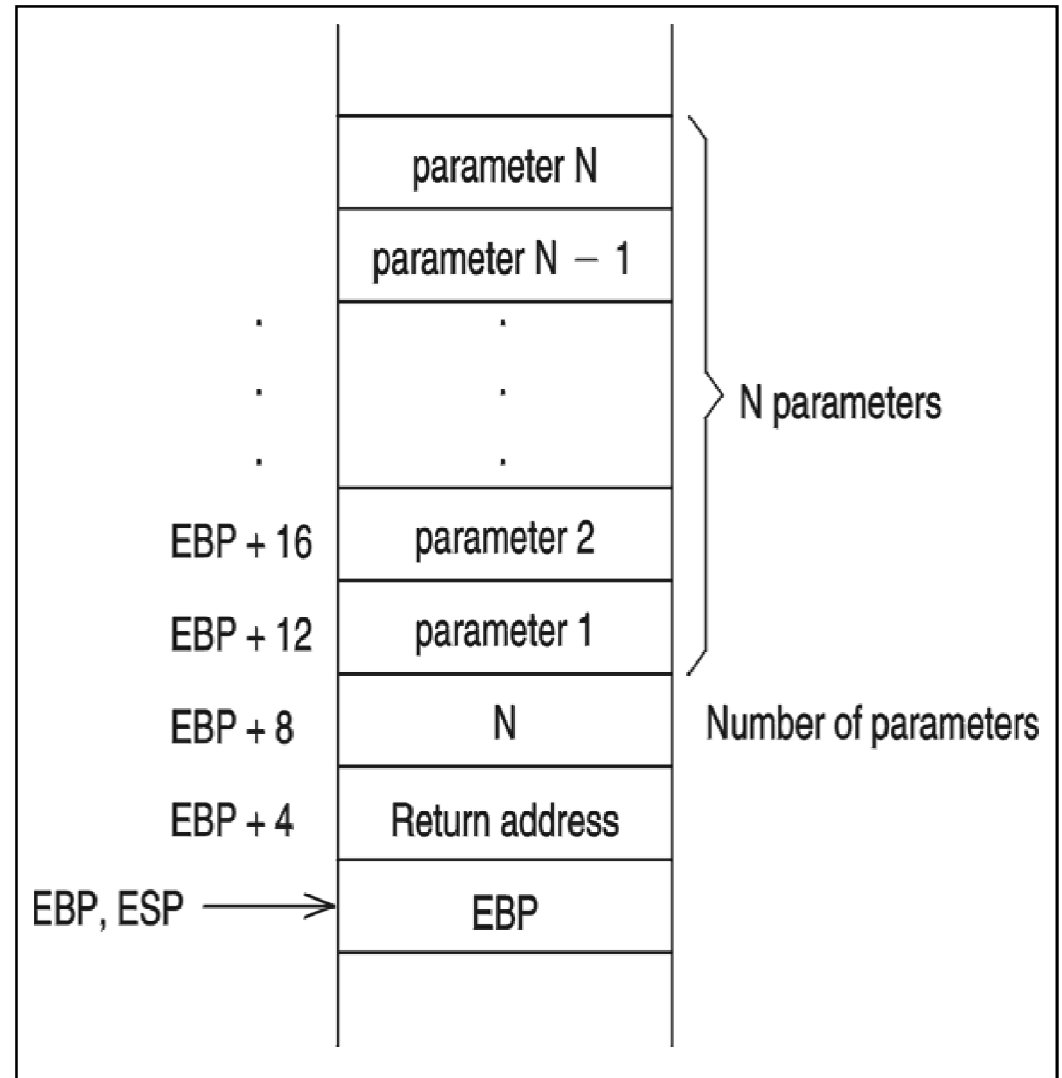
- * Implementează algoritmul de sortare prin metoda bulelor
- * utilizeaza **pusha** si **popa**

Număr variabil de parametri

- Cele mai multe proceduri au număr fix de parametri
 - * La fiecare apel același număr de parametri este transmis
- Proceduri cu număr variabil de parametri
 - * La fiecare apel numărul de parametri poate fi diferit
 - » C folosește acest tip de proceduri
 - * Ușor de implementat folosind transmiterea prin stivă

Număr variabil de parametri

- implementarea mecanismului de transmiterea unui număr variabil de parametri:
 - * Numărul de parametri trebuie să fie unul din parametri transmiși
 - * Acest număr trebuie să fie ultimul parametru pus pe stivă



Variabile locale

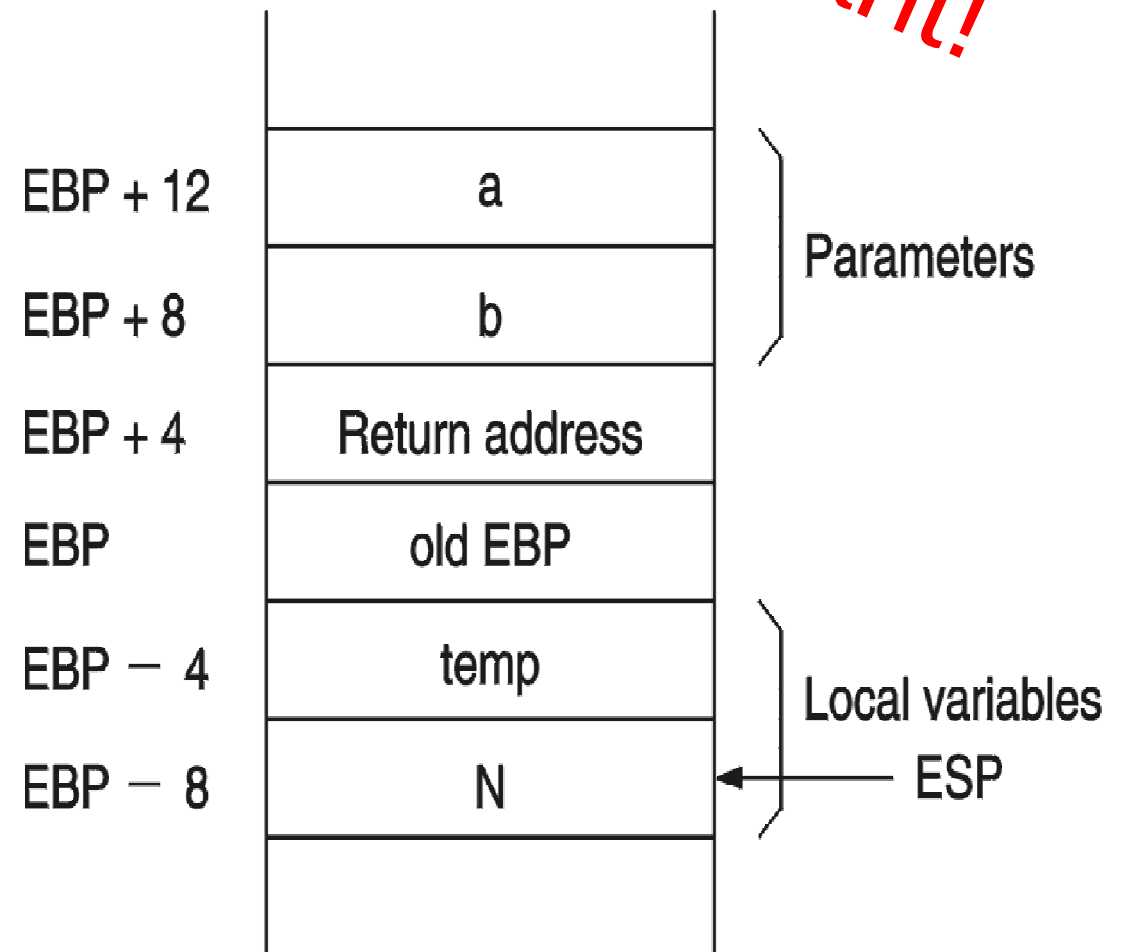
- Variabilele locale au natura dinamica
 - * Variabilele locale intra în existență la invocarea unei proceduri și se distrug odată cu terminarea execuției acesteia
- Nu se poate rezerva spațiu pentru aceste variabile în segmentul de date (.data) din două motive:
 - » Alocarea spațiului este statică (rămâne persistent între apelurile unei proceduri)
 - » Nu funcționează cu proceduri recursive
- Din aceste motive spațiul pentru variabile locale este rezervat pe stivă

Stack frame = activation record = cadru de stivă

```
void f(int b, int a){  
    int temp=1, N;  
    ...  
    return;  
}
```

```
call    f  
add     ESP, 8
```

```
f:  
    enter    8, 0  
    mov     [EBP-4],1  
    ...  
    leave  
    ret
```



Activation record

- Datele de pe stivă despre procedura curentă
 - » parametri
 - » adresa de retur
 - » vechiul EBP
 - » registre salvate
 - » variabile locale

} **cadru de stivă**
== **frame stack**
== **activation record**
- Fiecare apel de funcție necesită aceste informații
- EBP este denumit **base pointer**
 - * EBP cunoscut => acces la toate datele din stack frame
 - * Lista înlănțuită de stack frame-uri

Variabile locale – exemple

- **asm_ch5/procfib1.asm**

- * In cazul procedurilor simple, registrele pot fi folosite pentru stocarea variabilelor locale
- * Utilizarea registrelor pentru stocarea variabilelor locale
- * Afisarea celui mai mare numar Fibonacci mai mic decat un numar dat ca input

- **asm_ch5/procfib2.asm**

- * Foloseste stiva pentru stocarea variabilelor locale
- * Aspecte legate de performanta utilizarii registrelor vs stiva vor fi discutate ulterior

Performanță în apeluri de funcții

Stiva versus Registre ./asm_ch5/bbbsort.asm

- *Fara procedura swap (Program 5.5, lines 95-99)*

swap:

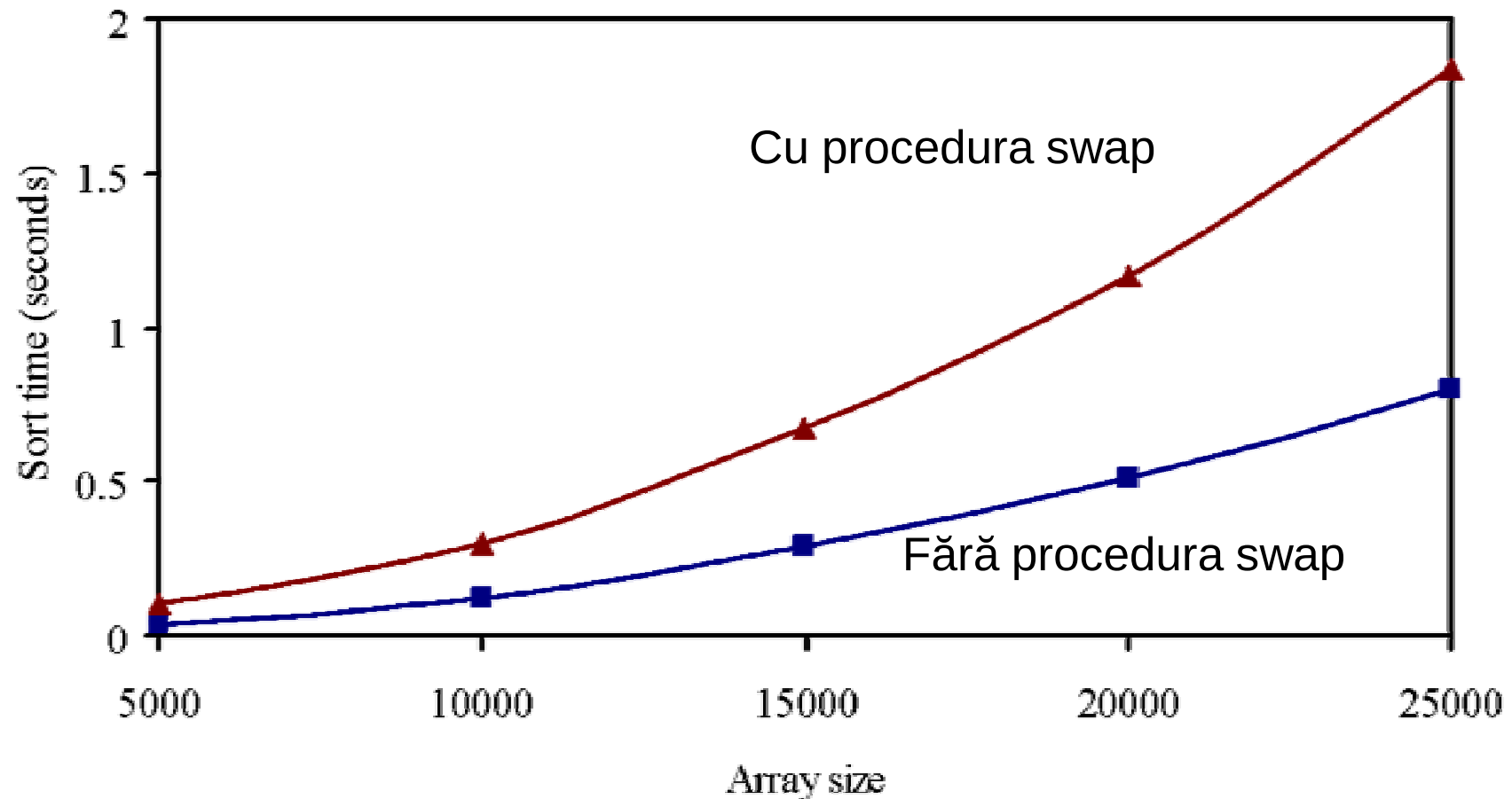
```
mov [ESI+4],EAX
mov [ESI],EBX
mov EDX,UNSORTED
```

- *Procedura SWAP (inlocuieste codul de mai sus)*

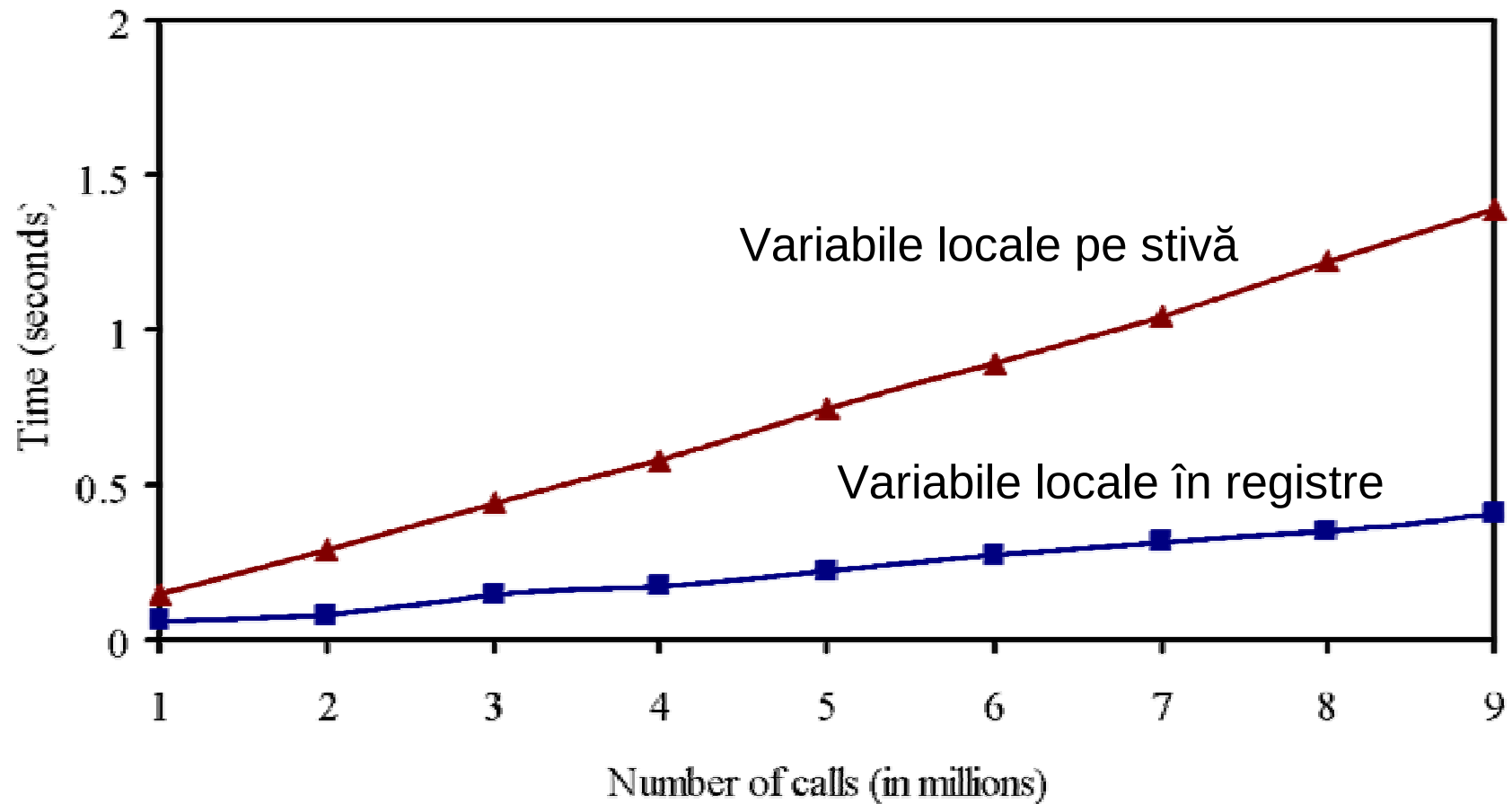
swap_proc:

```
mov [ESI+4],EAX
mov [ESI],EBX
mov EDX,UNSORTED
ret
```

Performanță în apeluri de funcții



Performanta: Overhead variabile locale



Recursivitate

Chapter 16

S. Dandamudi

Introducere

- O functie recursiva este o functie care se apelează pe sine însăși

- * Direct, sau indirect

- Unele aplicații pot fi exprimate în mod natural prin recursivitate

- $\text{factorial}(0) = 1$

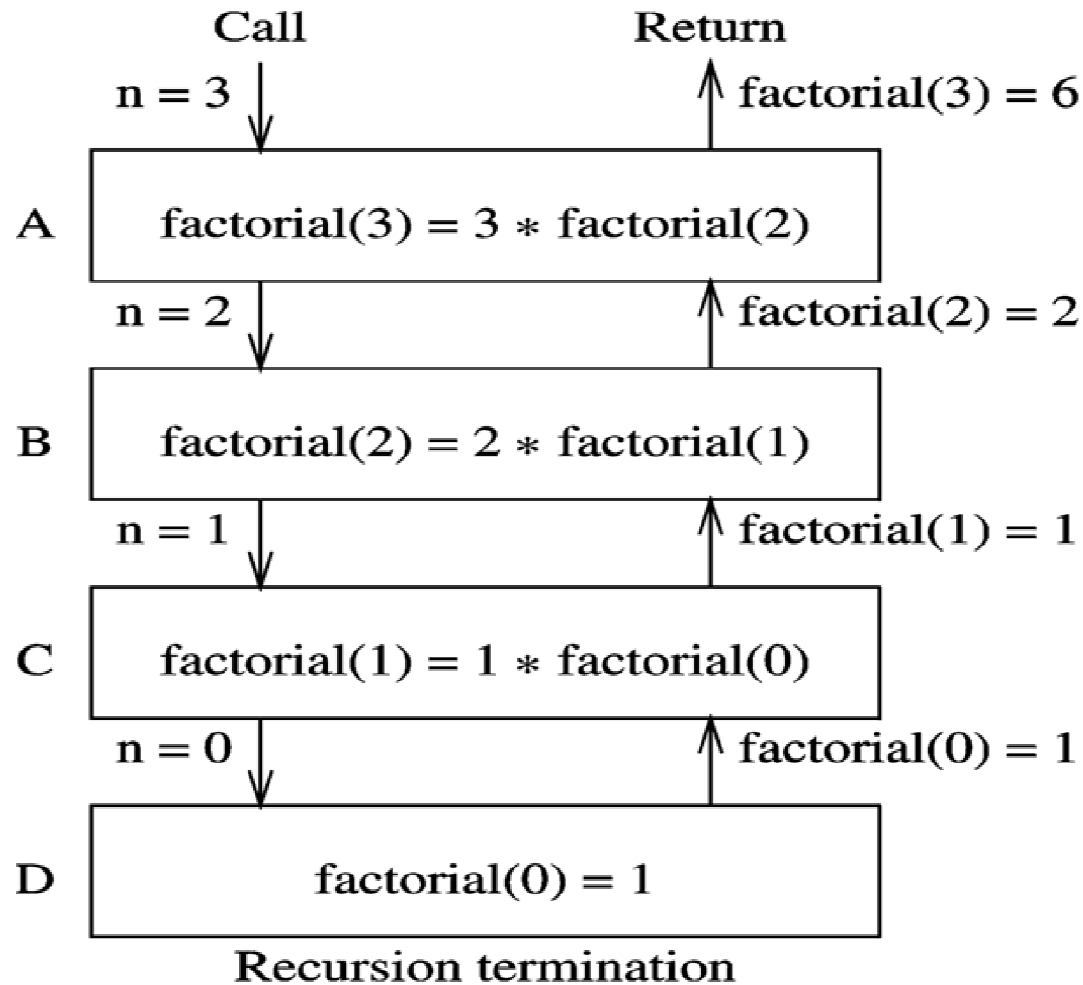
- $\text{factorial}(n) = n * \text{factorial}(n-1)$ for $n > 0$

- Din punct de vedere al implementării

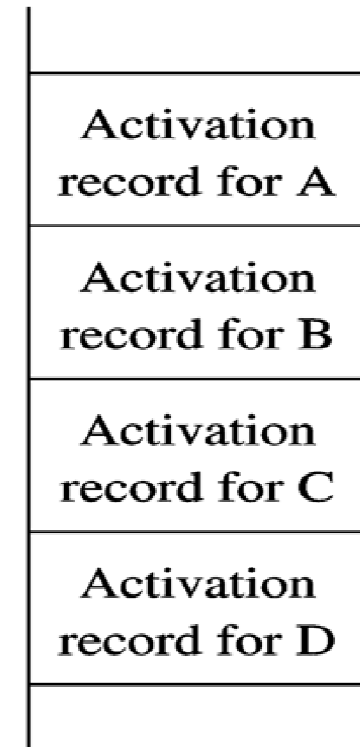
- Similar cu orice alt apel de functie

- La fiecare apel de functie se creeaza un stack frame

Introducere (cont'd)



(a)



(b)

Recurzivitate

- Doua exemple
 - * Factorial ./asm_ch16/fact_pentium.asm
 - * Quicksort ./asm_ch16/qsort_pentium.asm

Exemplu 1

- * Factorial

$\text{factorial}(0) = 1$

$\text{factorial}(n) = n * \text{factorial}(n-1)$ for $n > 0$

- **Activation record**
 - * Consta doar în stocarea adresei de retur pe stivă cu ajutorul instrucțiunii call
 - * Parametru în BL

Recursivitate

Exemplu 2

- Quicksort
 - * Sortarea unui vector de N elemente
 - * Algoritm
 - » Selectam un element pivot x
 - » Presupunem ca ultima aparitie a lui x este $\text{array}[i]$
 - » Mutam toate elementele mai mici decat x pe pozitiile $\text{array}[0] \dots \text{array}[i-1]$
 - » Mutam toate elementele mai mari decat x pe pozitiile $\text{array}[i+1] \dots \text{array}[N-1]$
 - » Aplicam quicksort recursiv pentru a sorta cele doua subliste

Recursiv vs Iterativ

- Recursiv
 - * Concis
 - * Mentenanta mai usoara a programului
 - * Alegerea naturala pentru unele probleme
- Posibile probleme
 - * Ineficient
 - » Apelurile recursive produc overhead
 - » Calcule duplicate
 - * Necesita mai multa memorie
 - * Cadre de stivă

Recursivitatea la coadă

- Tail recursion
- **Numai** când ultima instrucțiune este apelul recursiv
- se poate optimiza apelul recursiv ca un jmp
- nu se mai creează o activare pe stivă
- Exemplu ./tail_rec
- make && make asm
- Examinare fact.s și fib.s

Interfața cu limbajele de nivel înalt

Chapter 17
S. Dandamudi

Cuprins

- De ce programe mixte?
 - * **Focus pe C si limbaj de asamblare**
- Compilarea programelor mixte
- Apel limbaj de asamblare din C
 - * Transmiterea parametrilor
 - * Valori de retur
 - * Pastrarea valorilor din registre
 - * Global si external
- Exemple
- Apeluri C din limbaj de asamblare

De ce programe mixte?

- Avantaje si dezavantaje ale limbajului de asamblare
 - * **Avantaje:**
 - » Acces la operatii low-level
 - » Performanta
 - » Control asupra programului
 - * **Dezavantaje:**
 - » Productivitate scazuta
 - » Greu de asigurat mentenanta
 - » Lipsa de portabilitate
- Prin urmare, unele programe sunt mixte (system software)

Compilarea programelor mixte

- Putem folosi programare mixta in C si limbaj de asamblare
- Vom pune accentul pe principii
- Acestea pot fi generalizate la orice tip de programare mixta
- Pentru compilare:

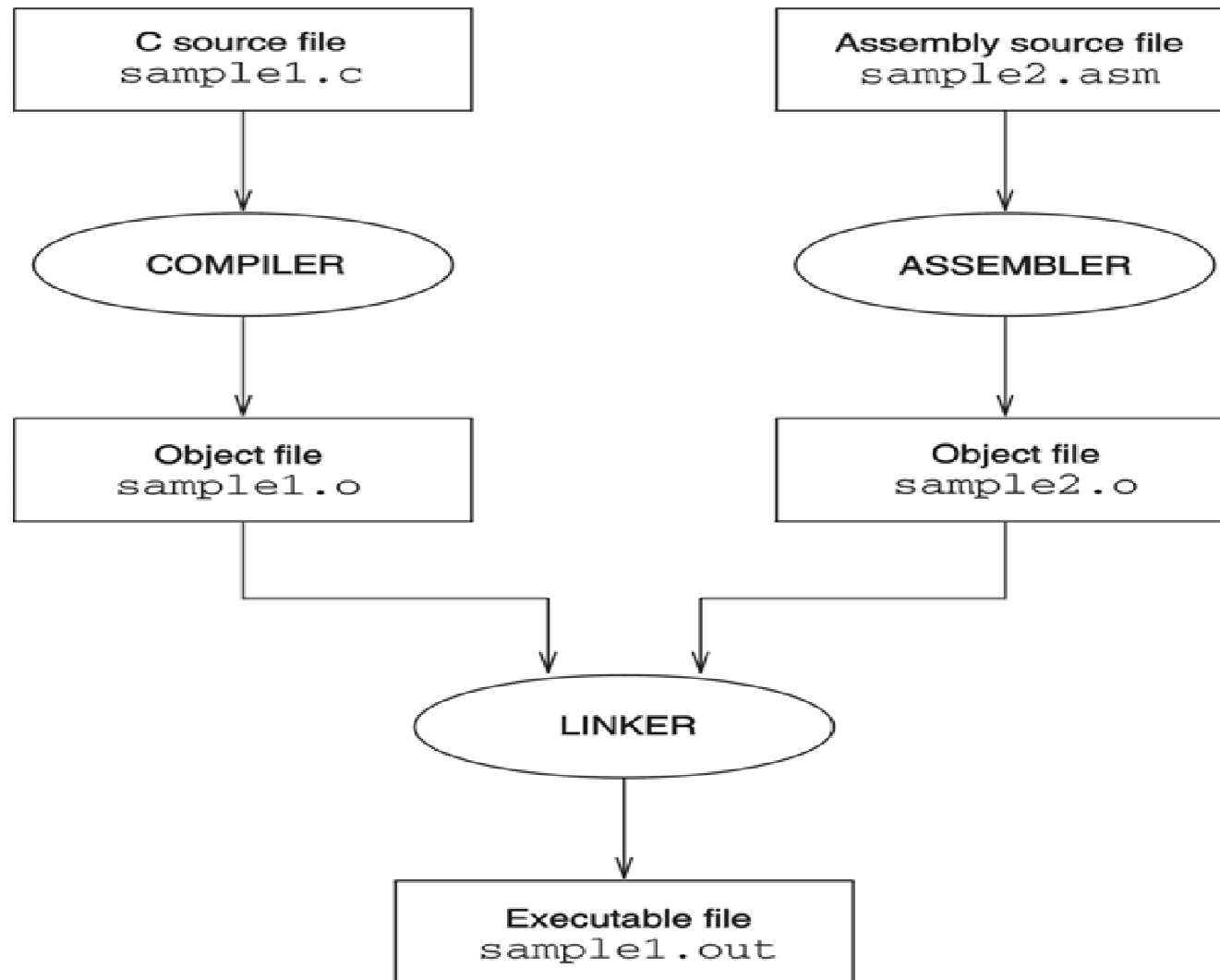
nasm -f elf sample2.asm

» creeaza **sample2.o**

gcc -o sample1.out sample1.c sample2.o

» creeaza **sample1.out** fisier executabil

Obținerea unui executabil mixt



Apel limbaj de asamblare

Transmiterea parametrilor

- Stiva este folosita pentru transmiterea parametrilor
- Doua modalitati de a pune parametrii pe stiva
 - * **Left-to-right**
 - » Majoritatea limbajelor inclusiv Basic, Fortran, Pascal folosesc aceasta metoda
 - » Aceste limbaje se numesc limbaje *left-pusher*
 - * **Right-to-left**
 - » **C foloseste aceasta metoda**
 - » Aceste limbaje se numesc limbaje *right-pusher*

Apel limbaj de asamblare din C

Exemplu:

sum(a,b,c,d)



Apel limbaj de asamblare din C

Valoare de retur

- Registrele folosite pentru valori de retur

Tip valoare return	Registru folosit
8-, 16-, 32-bit value	EAX
64-bit value	EDX:EAX

- Valorile reprezentate in virgula mobila sunt discutate in capitolul urmator

Apel limbaj de asamblare din C

Pastrarea valorilor registrelor

- Valorile urmatoarelor registre trebuie pastrate

EBP, EBX, ESI, si EDI

- Alte registre
 - * Daca este necesar, valorile pot fi pastrate de catre functia apelanta

Apel limbaj de asamblare din C

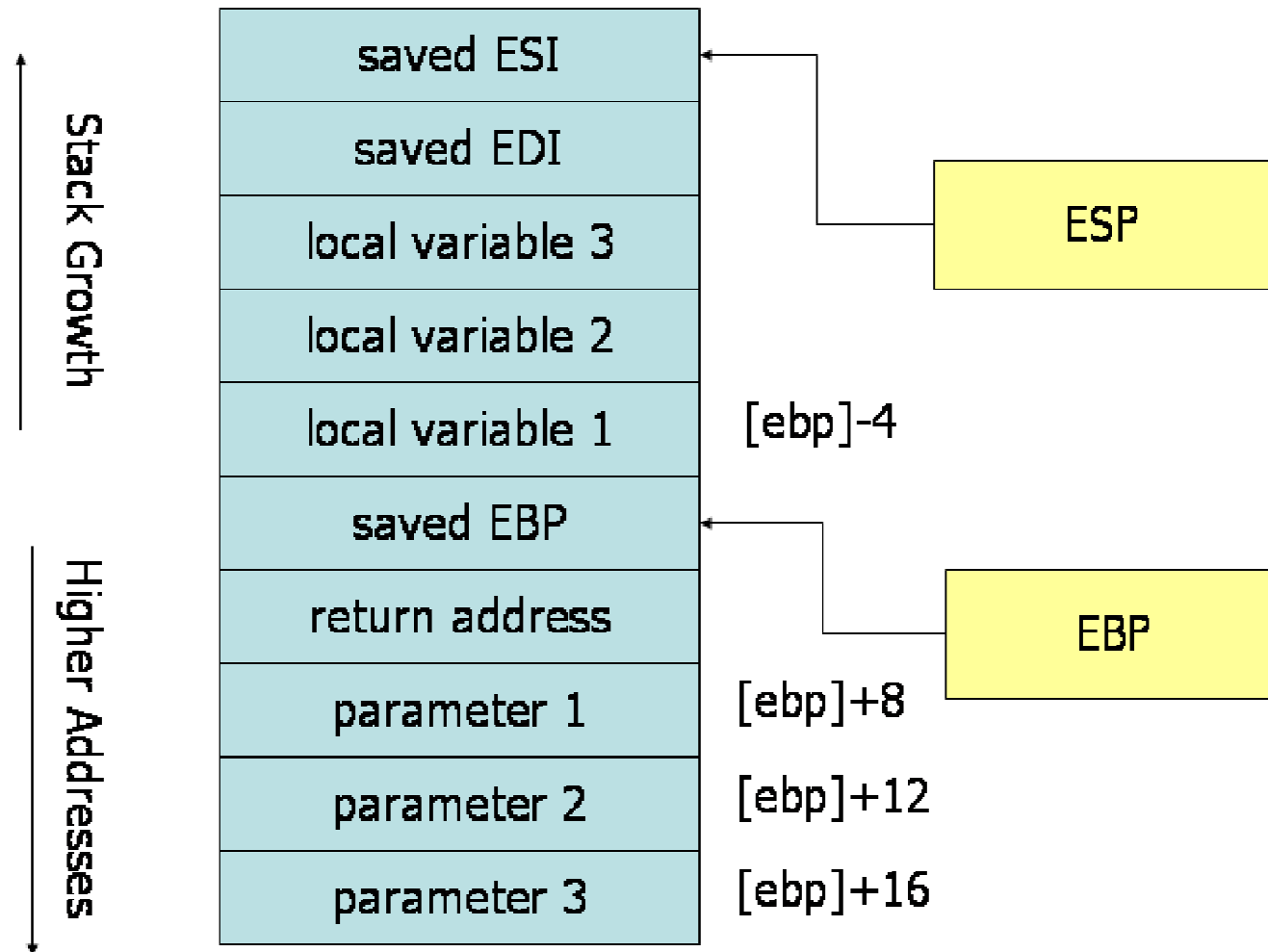
Global si External

- Programarea mixta implica cel putin 2 module
 - » Un modul in C si un modul in limbaj de asamblare
- Trebuie sa declaram functiile si procedurile care nu sunt definite in acelasi modul ca fiind external
 - » Section 5.10
- Procedurile folosite de alte module sunt globale

Exemple

- Exemplu 1 în ./asm_ch17
 - * **hll_ex1c.c**
 - * **hll_test.asm**
- Exemplu 2
 - * **hll_minmaxc.c**
 - * **hll_minmaxa.asm**
- Exemplu 3
 - * **hll_arraysumc.c**
 - * **hll_arraysuma.asm**

x86 cdecl calling convention (2)



<http://www.cs.virginia.edu/~evans/cs216/guides/stack-convention.png>

Apelarea unei funcții cu parametri

push paramN

push paramN-1

....

push param2

push param1

call func

Restaurarea stivei după apelul unei funcții

push paramN

push paramN-1

....

push param2

push param1

call func

add esp, N*4

Salvarea registrelor

- Unele registre pot modificate în cadrul unei funcții
- Dacă avem nevoie de valori înainte sau după trebuie salvate în prealabil
 - * Pe stivă (push eax), cel mai comun
 - * Într-o zonă de memorie (date)
- După apelul funcției registrele în cauză sunt restaurate (pop de pe stivă, de exemplu)

Referirea parametrilor unei funcții

- [EBP]: saved ebp
- [EBP+4]: return address
- [EBP+8]: first parameter
- [EBP+12]: second parameter
- [EBP+16]: third parameter
- ...

Apeluri C din limbaj de asamblare

CDECL Sumar

- Apelantul
 - * Plasează parametrii pe stivă (in ordine inversa)
 - * conservă EAX, ECX, EDX dacă este cazul
- Apelatul
 - * Alocă și eliberează variabilele locale
 - * pastreaza valorile registrelor EBP, EBX, ESI, si EDI (callee preserved)
 - * Valoarea de retur este stocata in EAX, sau EDX:EAX
- Apelantul
 - * Curăță stivei **după retur**
add ESP,4

Inline Assembly

- Cod în limbaj de asamblare integrat în codul C
 - » Nu este necesar un modul separat scris în limbaj de asamblare
- Construcțiile în limbaj de asamblare sunt identificate cu ajutorul cuvântului cheie **asm**
- Putem folosi () pentru a grupa mai multe construcții în limbaj de asamblare

```
asm(  
    assembly statement  
    assembly statement  
    . . .  
);
```

Inline Assembly (cont'd)

- Exemple Inline

- * Exemplu 1

- » **hll_ex1_inline.c**

- * Exemplu 2

- » Suma elementelor unui vector

- » **hll_arraysum_inline.c**

- * Exemplu 3

- » A doua versiune a ultimului exemplu

- » **hll_arraysum_inline2.c**