



Bitdefender[®]. Awake.

Programare multi-modul

Marius Vanța
mvanta@bitdefender.com

3. Interfațarea cu limbajele de nivel înalt

Interfațarea cu limbajele de nivel înalt



- Transmiterea parametrilor **prin valoare**
 - Se copiază valorile octeților ce compun datele
 - Risipitor și lent când datele ocupă multă memorie!
 - Din perspectiva apelantului, datele sunt constante!
- Transmiterea parametrilor **prin referință** (adresă/pointer)
 - Se specifică adresa (și uneori dimensiunea) datelor
 - Ineficient când datele ocupă puțin
 - 32 biți în plus pentru stocat adresa + citire pointer înainte de acces la date
 - Datele pot suferi modificări
- Decizia transmitere prin valoare sau prin referință
 - Criteriul de performanță: dimensiunea în octeți
 - Risc de a depăși memoria disponibilă? Prin referință!
 - Criteriul de accesibilitate: trebuie modificate datele?
 - Neapărat constante? Prin valoare!

- Convenții de apel (call conventions)
 - Cum transmitem parametri către subrutine?
 - Ce tipuri de parametri se pot transmite?
 - În ce ordine?
 - Câți parametri? Oricâți?
 - Ce resurse sunt volatile (poate să le altereze funcția apelată)?
 - Unde se regăsește rezultatul?
 - Ce acțiuni de curățare (cleanup) sunt necesare post-apel?
 - Cine este responsabil să le efectueze?
 - Convenții
 - Uzuale: **CDECL**, **STDCALL**, FASTCALL
 - Rar folosite sau învechite: PASCAL, FORTRAN, SYSCALL, etc...
 - Utilizator: **în asamblare toate aspectele documentate de către o convenție sunt accesibile programatorului!**

Interfațarea cu limbajele de nivel înalt



- Convenții de apel – convenția C (**CDECL**)
 - Specifică limbajului C
 - Cum transmitem parametri către subrutine? Prin împingerea lor pe stivă
 - Ce tipuri de parametri se pot transmite? Orice, dar necesită extins la minim DWORD
 - În ce ordine? Dreapta către stânga, adică, invers ordinii de la declarație
 - Câți parametri? **Oricâți**? Da, C permite funcții cu oricâți parametri (ex: printf)
 - Ce resurse sunt volatile? **EAX, ECX, EDX, Eflags**
 - Unde se regăsește rezultatul? EAX, EDX:EAX sau ST0 (FPU)
 - Ce acțiuni de curățare (cleanup) sunt necesare? Eliberarea argumentelor
 - Cine este responsabil să le efectueze ? **Apelantul!**

Parametri			Resurse volatile	Rezultate	Eliberare
Stocare	Ordine	Număr			
Stivă	Inversă	<u>Oricâți</u>	EAX, ECX, EDX, Eflags	EAX / EDX:EAX / ST0 (FPU)	<u>Apelant</u>

Interfațarea cu limbajele de nivel înalt



- Convenții de apel – convenția **STDCALL**
 - Specifică sistemului de operare Windows
 - Denumită și **WINAPI**
 - Folosită de către bibliotecile de sistem Windows
 - Foarte asemănătoare convenției CDECL
 - Diferențe:
 - Număr *fix* de parametri
 - Eliberarea argumentelor o face *funcția apelată*

Parametri			Resurse volatile	Rezultate	Eliberare
Stocare	Ordine	Număr	EAX, ECX, EDX, EFlags	EAX / EDX:EAX / ST0 (FPU)	<u>Subrutina apelată</u>
Stivă	Inversă	<u>Fix</u>			

Interfațarea cu limbajele de nivel înalt



- Apelul subrutinelor

- Etape:

1. **Cod de apel**: pregătirea și efectuarea apelului
2. **Cod de intrare**: intrarea în procedură și pregătirea execuției
3. **Cod de ieșire**: revenire și eliberarea resurselor ce au expirat

- Acțiunile depind în funcție de convenția de apel a subrutinei apelate – dar etapele rămân aceleași!
- Etapele sunt tratate/implementate automat în codul generat de către compilatoarele limbajele de nivel mai înalt
 - În asamblare rămâne totul în sarcina noastră!

Interfațarea cu limbajele de nivel înalt



- Apelul subrutinelor – cod de apel
 - Sarcini:
 1. Salvare resurse volatile în uz: *push registru*
 2. Asigurare respectare constrângeri (ESP aliniat, DF=0, ...)
 3. Pregătire argumente (stivă, conform convenției): *push*
 4. Efectuare apel: *call*
 - *call subrutină când subrutină este linkeditată static*
 - *call [subrutină] dacă este dinamică (la link-time)*
 - *call registru sau call [variabilă] pentru dinamică la runtime*
- Subrutinele asm folosite doar din asm pot evita (din simplitate si/sau eficientă) aceste sarcini

Interfațarea cu limbajele de nivel înalt



- Apelul subrutinelor – cod de apel
 - Exemplu: apel printf din asm pentru afișare numere 0..9

```
import exit msvcrt.dll
import printf msvcrt.dll
extern exit, printf
global start
```

```
segment code use32
start:
    mov ecx, 10
    xor eax, eax
```

```
.next:
```

```
    push eax
    push ecx
```

```
    push eax
```

```
    push dword format_string
```

```
    call [printf]
```

```
    add esp, 2*4
```

```
    pop ecx
```

```
    pop eax
```

```
    inc eax
```

```
loop .next
```

```
push dword 0
```

```
call [exit]
```

```
segment data use32
```

```
    format_string db "%d", 10, 13, 0
```

Dacă apelul se făcea din C, compilatorul ar fi generat singur codul de apel!
Cum apelul se face din asamblare, codul de apel trebuie scris de către noi!

- (1) Salvare resurse volatile
 - (2) DF=0, stiva a fost folosită doar pe DWORD
 - (3) Pregătire argumente apel CDECL
 - (4) Efectuare apel
- Restaurare resurse volatile (apelantul)

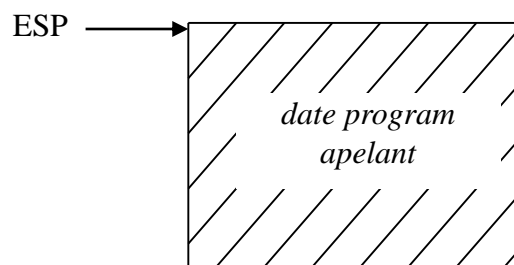
Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de apel
 - Efectul codului de apel asupra stivei

```
push eax  
push ecx
```

```
push eax  
push dword format_string  
call [printf]  
add esp, 2*4
```

Stare inițială



Interfațarea cu limbajele de nivel înalt

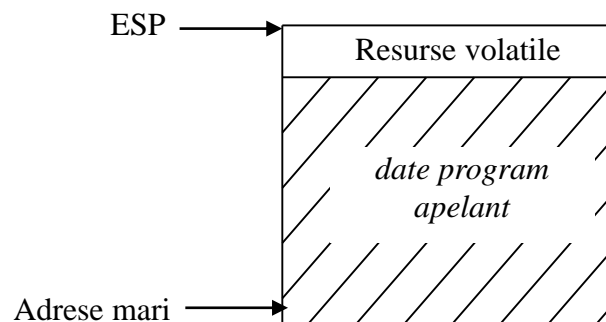
- Apelul subrutinelor – cod de apel
 - Efectul codului de apel asupra stivei

```
push eax  
push ecx
```

→ (1) Salvare resurse volatile

```
push eax  
push dword format_string  
call [printf]  
add esp, 2*4
```

(1) Salvare resurse volatile



Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de apel
 - Efectul codului de apel asupra stivei

```
push eax  
push ecx
```

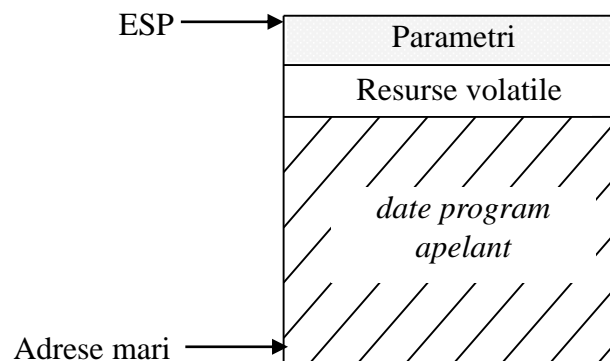
```
push eax  
push dword format_string  
call [printf]  
add esp, 2*4
```

(1) Salvare resurse volatile

(2) DF=0, stiva a fost folosită doar pe DWORD

(3) Pregătire argumente apel CDECL

(3) Pregătire argumente apel CDECL



Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de apel

- Efectul codului de apel asupra stivei

```
push eax  
push ecx
```

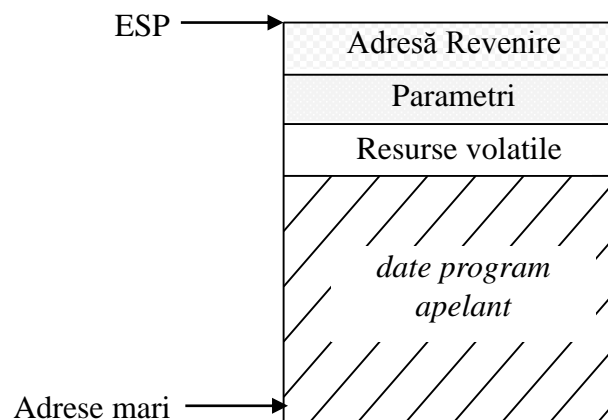
```
push eax  
push dword format_string
```

```
call [printf]
```

```
add esp, 2*4
```

- (1) Salvare resurse volatile
- (2) DF=0, stiva a fost folosită doar pe DWORD
- (3) Pregătire argumente apel CDECL
- (4) Efectuare apel

(4) Efectuare apel



- Apelul subrutinelor – cod de intrare
 - Sarcini:
 1. Configurarea unui **cadru de stivă** (stack frame): reper ebp sau esp?
 2. Pregătire variabile locale ale funcției: sub esp, număr_octeti
 3. Salvarea unei copii a resurselor *nevolatile* modificate: push registru
 - *Orice regiștri cu excepția celor volatili*
 - Cadru de stivă: structură de date stocată în stivă, de dimensiune fixă (pentru o subrutină dată) și conținând:
 - *Parametrii pregătiți de apelant*
 - *Adresa de revenire (către instrucțiunea ce-i urmează celei de apel)*
 - *Copii ale resurselor nevolatile folosite de subrutină*
 - *Variabile locale*
 - Subrutinele asm folosite doar din asm pot evita (din simplitate sau eficientă) aceste sarcini

Interfațarea cu limbajele de nivel înalt



- Apelul subrutinelor – cod de intrare

- Exemplu: cod intrare în funcția CDECL **chdir**, generată de compilatorul C

```
; dezasamblat cu IDA - The Interactive Disassembler  
; sintaxa corespunde asamblorului MASM  
; int __cdecl chdir(const char *)  
chdir:
```

(1) Configurare stack frame	→	mov edi, edi ; inutil, dar permite modificare!
(2) Variabile locale	→	push ebp
(3) Salvare regiștri nevolatili	→	mov ebp, esp
	→	sub esp, 118h
	→	mov eax, __security_cookie
	→	xor eax, ebp
	→	mov [ebp+var_4], eax
	→	mov eax, [ebp+lpPathName]
	→	and [ebp+var_110], 0
	→	push ebx
	→	or ebx, 0FFFFFFFFh
	→	push esi
		;

Codul de intrare a fost generat automat de către compilator!

Interfațarea cu limbajele de nivel înalt



- Apelul subrutinelor – cod de intrare

- Exemplu: cod intrare într-o funcție STDCALL scrisă în asm de către noi

	factorial://in modulul C apelant se declara - extern int _stdcall factorial(int n)	
(1) Configurare stack frame	<code>push ebp</code>	
	<code>mov ebp, esp</code>	; definim un cadru de stiva cu EBP pivot/referinta
(2) Variabile locale	<code>sub esp, 4</code>	; alocam spatiu pe stiva pentru o variabila DWORD temporara
	<code>push ebx</code>	; salvam ebx pentru a-l putea restaura la sfarsit
(3) Salvare regiștri nevolatili		
	<code>mov eax, [ebp + 8]</code>	; incarcam valoarea argumentului din stackframe (n-ul curent)
	<code>cmp eax, 2</code>	; testam coditia de terminare (n < 2)
	<code>jae .recursiv</code>	
	<code>mov eax, 1</code>	; returnam 1 cand n < 2
	<code>jmp .gata</code>	
int factorial (int n)		
{ if (n==1) return 1;	<code>.recursiv:</code>	
else	<code>push eax</code>	; retinem valoarea lui n sa nu-l pierdem la apelul recursiv
return n * factorial (n-1);	<code>dec eax</code>	; pregatim noul parametru n-1 pt a apela factorial(n-1)
}	<code>push eax</code>	; urcam in stiva valoarea n-1 ca parametru pt factorial(n-1)
	<code>call factorial</code>	; apel recursiv (STDCALL)
	<code>mov [ebp - 4], eax</code>	; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
	<code>pop eax</code>	; restauram valoarea lui n
	<code>mov ebx, [ebp - 4]</code>	; ebx <- factorial(n-1), preluat din variabila temporara
	<code>mul ebx</code>	; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)
	<code>.gata:</code>	
	<code>pop ebx</code>	; refacem EBX la valoarea initiala
	<code>add esp, 4</code>	; eliberam memoria ocupata de variabila temporara
	<code>pop ebp</code>	; restauram ebp la valoarea initiala
	<code>ret 4</code>	; STDCALL: revenire cu eliberare memorie parametri

Interfațarea cu limbajele de nivel înalt



- Apelul subrutinelor – cod de intrare

- Exemplu: cod intrare într-o funcție STDCALL scrisă în asm – stackframe

```
factorial:
    push ebp                ; definim un cadru de stiva cu EBP pivot/referinta
    mov ebp, esp            ; alocam spatiu pe stiva pentru o variabila DWORD temporara
    sub esp, 4              ; salvam ebx pentru a-l putea restaura
    push ebx

    mov eax, [ebp + 8]      ; incarcam valoarea argumentului din stackframe

    cmp eax, 2              ; testam coditia de terminare (n < 2)
    jae .recursiv
    mov eax, 1              ; returnam 1 cand n < 2
    jmp .gata

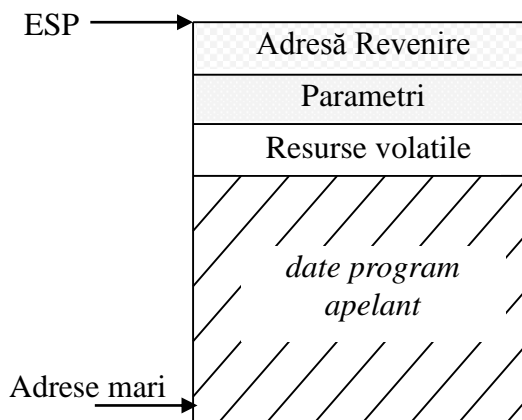
.recursiv:
    push eax                ; retinem valoarea lui n sa nu-l pierdem la apelul recursiv

    dec eax
    push eax                ; apelam factorial(n-1), am urcat in stiva valoarea n-1
    call factorial          ; apel recursiv (STDCALL)

    mov [ebp - 4], eax       ; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
    pop eax                 ; restauram valoarea lui n
    mov ebx, [ebp - 4]       ; ebx <- factorial(n-1), preluat din variabila temporara
    mul ebx                 ; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)

.gata:
    pop ebx                 ; refacem EBX la valoarea initiala
    add esp, 4              ; eliberam memoria ocupata de variabila temporara
    pop ebp                 ; restauram ebp la valoarea initiala
    ret 4                   ; STDCALL: revenire cu eliberare memorie parametri
```

Stare la intrare



Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de intrare

- Exemplu: cod intrare într-o funcție STDCALL scrisă în asm – stackframe

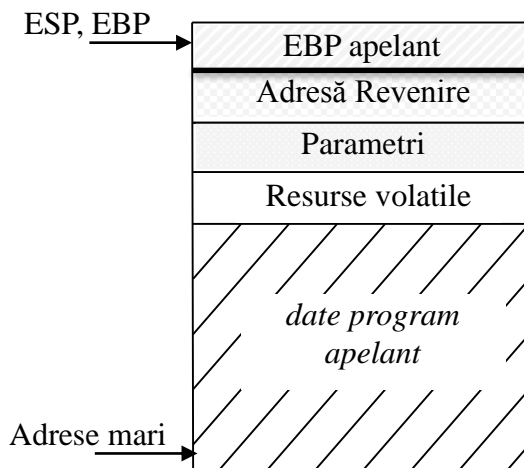
(1) Configurare stack frame →

```
factorial:
    push ebp                ; definim un cadru de stiva cu EBP pivot/referinta
    mov ebp, esp            ; alocam spatiu pe stiva pentru o variabila DWORD temporara
    sub esp, 4              ; salvam ebx pentru a-l putea restaura
    push ebx

    mov eax, [ebp + 8]      ; incarcam valoarea argumentului din stackframe

    cmp eax, 2              ; testam coditia de terminare (n < 2)
    jae .recursiv
    mov eax, 1              ; returnam 1 cand n < 2
    jmp .gata
```

(1) Configurare stack frame



```
.recursiv:
    push eax                ; retinem valoarea lui n sa nu-l pierdem la apelul recursiv

    dec eax
    push eax                ; apelam factorial(n-1), am urcat in stiva valoarea n-1
    call factorial          ; apel recursiv (STDCALL)

    mov [ebp - 4], eax       ; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
    pop eax                 ; restauram valoarea lui n
    mov ebx, [ebp - 4]       ; ebx <- factorial(n-1), preluat din variabila temporara
    mul ebx                 ; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)

.gata:
    pop ebx                 ; refacem EBX la valoarea initiala
    add esp, 4              ; eliberam memoria ocupata de variabila temporara
    pop ebp                 ; restauram ebp la valoarea initiala
    ret 4                   ; STDCALL: revenire cu eliberare memorie parametri
```

Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de intrare

- Exemplu: cod intrare într-o funcție STDCALL scrisă în asm – stackframe

(1) Configurare stack frame →
(2) Variabile locale →

```
factorial:
    push ebp
    mov ebp, esp
    sub esp, 4
    push ebx

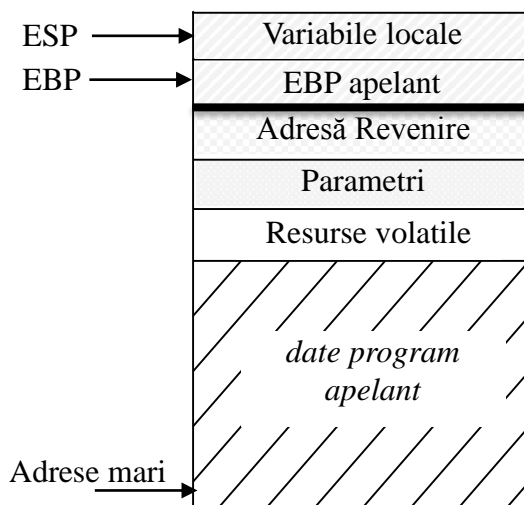
    mov eax, [ebp + 8]
    cmp eax, 2
    jae .recursiv
    mov eax, 1
    jmp .gata

    .recursiv:
    push eax
    dec eax
    push eax
    call factorial
    mov [ebp - 4], eax
    pop eax
    mov ebx, [ebp - 4]
    mul ebx

    .gata:
    pop ebx
    add esp, 4
    pop ebp
    ret 4
```

; definim un cadru de stiva cu EBP pivot/referinta
; alocam spatiu pe stiva pentru o variabila DWORD temporara
; salvam ebx pentru a-l putea restaura
; incarcam valoarea argumentului din stackframe
; testam coditia de terminare (n < 2)
; returnam 1 cand n < 2
; retinem valoarea lui n sa nu-l pierdem la apelul recursiv
; apelam factorial(n-1), am urcat in stiva valoarea n-1
; apel recursiv (STDCALL)
; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
; restauram valoarea lui n
; ebx <- factorial(n-1), preluat din variabila temporara
; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)
; refacem EBX la valoarea initiala
; eliberam memoria ocupata de variabila temporara
; restauram ebp la valoarea initiala
; STDCALL: revenire cu eliberare memorie parametri

(2) Variabile locale

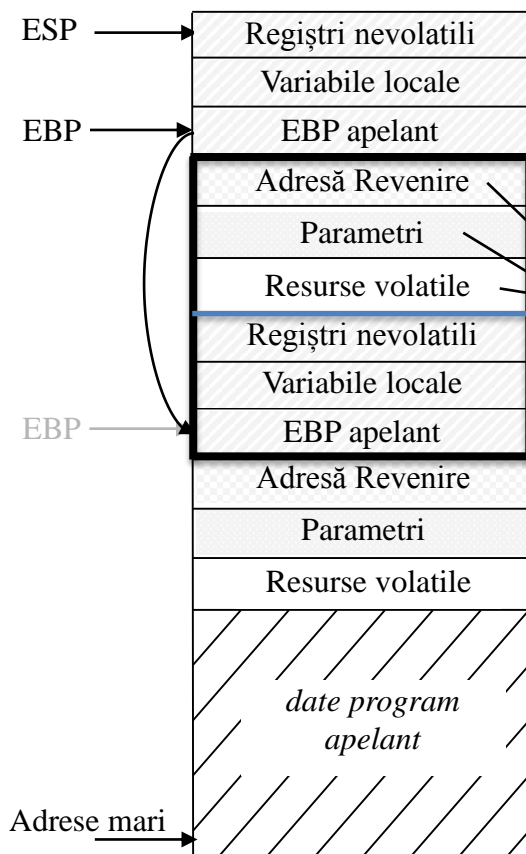


Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de intrare

- Exemplu: cod intrare într-o funcție STDCALL scrisă în asm – stackframe

Evoluție stivă la apel recursiv



```
factorial:
    push ebp
    mov ebp, esp           ; definim un cadru de stiva cu EBP pivot/referinta
    sub esp, 4             ; alocam spatiu pe stiva pentru o variabila DWORD temporara
    push ebx               ; salvam ebx pentru a-l putea restaura

    mov eax, [ebp + 8]     ; incarcam valoarea argumentului din stackframe

    cmp eax, 2             ; testam coditia de terminare (n < 2)
    jae .recursiv          ; returnam 1 cand n < 2
    mov eax, 1
    jmp .gata

    .recursiv:
    push eax               ; retinem valoarea lui n sa nu-l pierdem la apelul recursiv
    AICI SE GENEREAZA CODUL DE APEL PENTRU URMATORUL APEL RECURSIV !!
    dec eax
    push eax               ; apeland factorial(n-1), am urcat in stiva valoarea n-1
    call factorial          ; apel recursiv (STDCALL)

    mov [ebp - 4], eax     ; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
    pop eax                ; restauram valoarea lui n
    mov ebx, [ebp - 4]     ; ebx <- factorial(n-1), preluat din variabila temporara
    mul ebx                ; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)

    .gata:
    pop ebx                ; refacem EBX la valoarea initiala
    add esp, 4             ; eliberam memoria ocupata de variabila temporara
    pop ebp                ; restauram ebp la valoarea initiala
    ret 4                  ; STDCALL: revenire cu eliberare memorie parametri
```

Interfațarea cu limbajele de nivel înalt

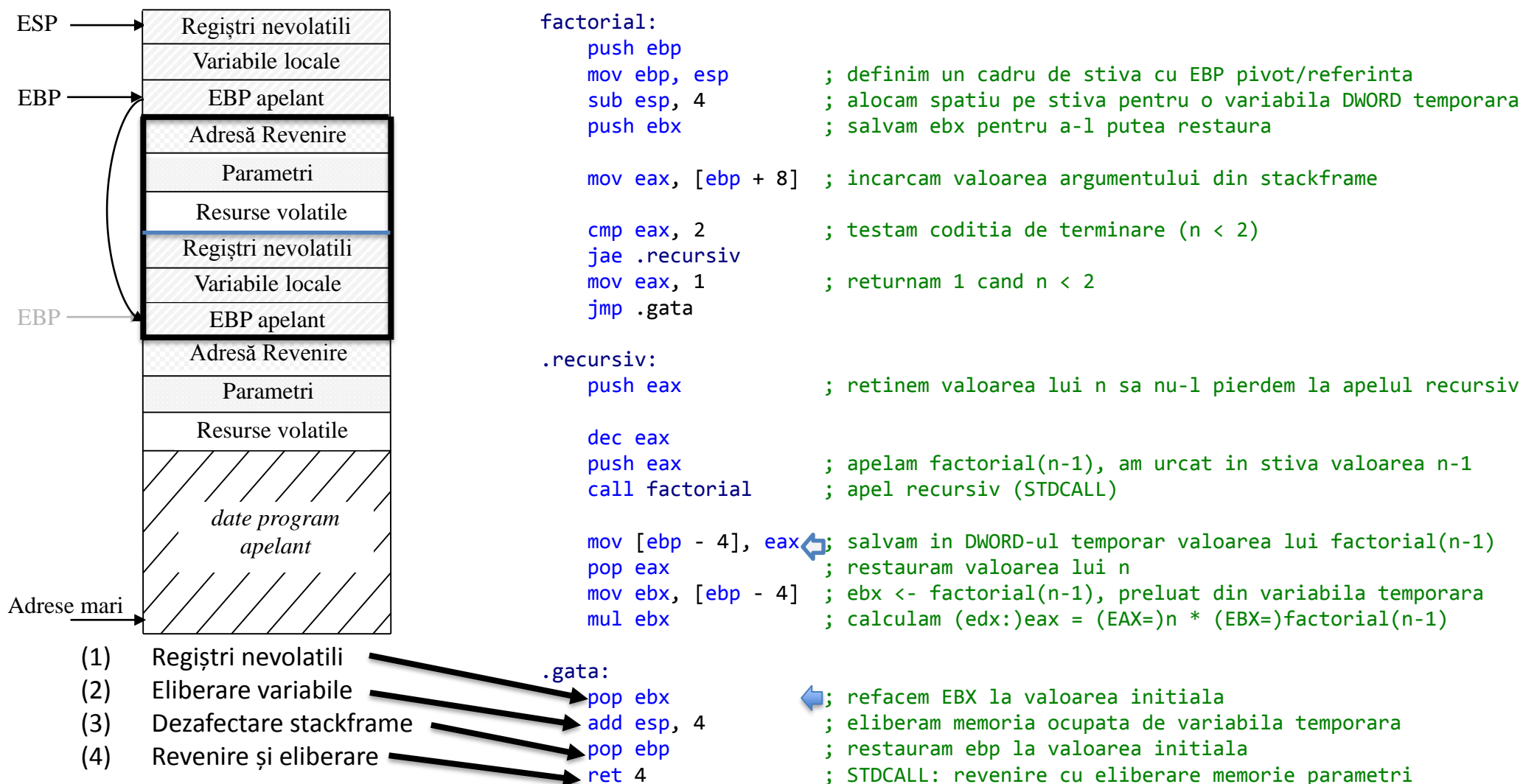


- Apelul subrutinelor – cod de ieșire
 - Sarcini:
 1. Restaurare resurse nevolatile alterate
 2. Eliberarea variabilelor locale ale funcției
 3. Dezafectarea cadrului de stivă
 4. Revenirea din funcție și eliberarea argumentelor
 - *CDECL:*
 - Subrutina apelată: ret
 - Subprogramul apelant: add esp, dimensiune_argumente
 - *STDCALL:*
 - *ret dimensiune_argumente*
 - Exceptând resursele volatile și rezultatele directe ale funcției, **starea programului după acești pași trebuie să reflecte starea inițială, de dinainte de apel!**

Interfațarea cu limbajele de nivel înalt

• Apelul subrutinelor – cod de ieșire

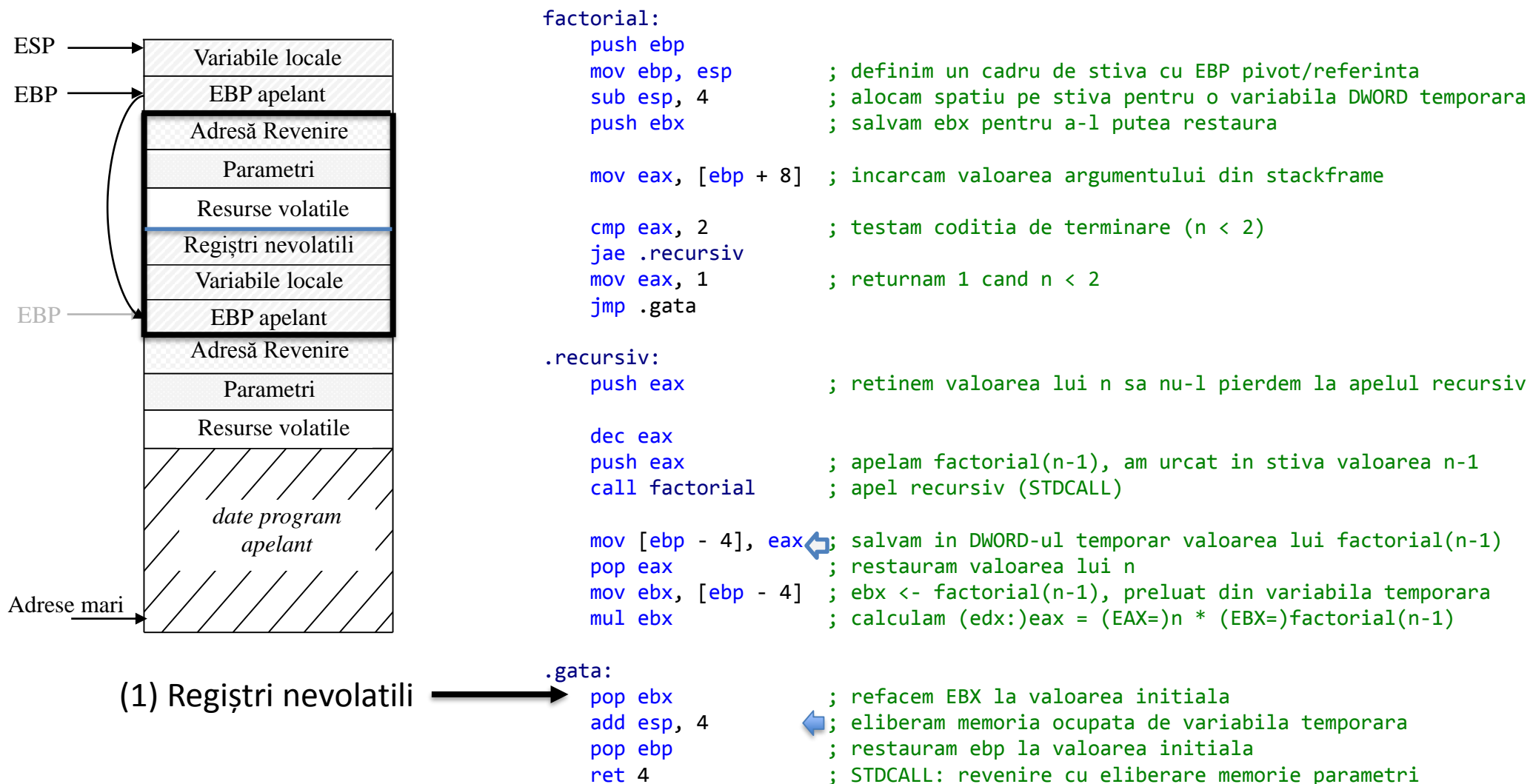
- Exemplu: cod ieșire dintr-o funcție STDCALL asm (din apel recursiv) - stackframe



Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de ieșire

- Exemplu: cod ieșire dintr-o funcție STDCALL asm (din apel recursiv) - stackframe

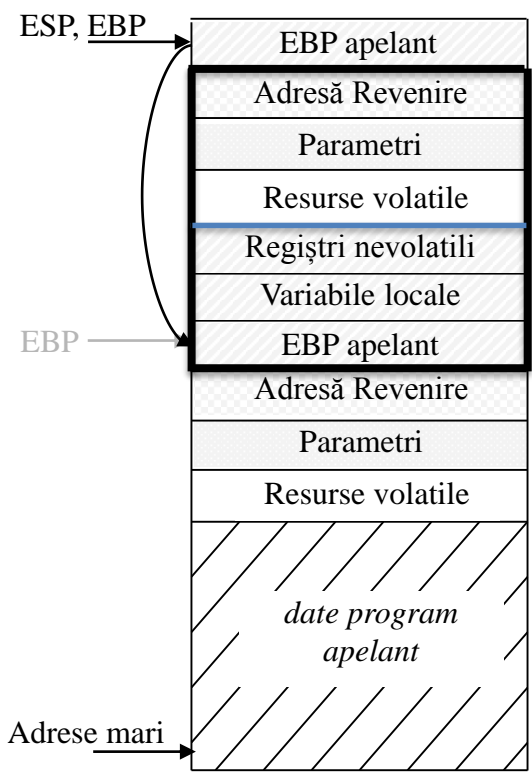


Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de ieșire

- Exemplu: cod ieșire dintr-o funcție STDCALL asm (din apel recursiv) - stackframe

Diagrama stackframe-ului:



```
factorial:
    push ebp
    mov ebp, esp
    sub esp, 4
    push ebx

    mov eax, [ebp + 8]
    cmp eax, 2
    jae .recursiv
    mov eax, 1
    jmp .gata

.recursiv:
    push eax
    dec eax
    push eax
    call factorial
    mov [ebp - 4], eax
    pop eax
    mov ebx, [ebp - 4]
    mul ebx

.gata:
    pop ebx
    add esp, 4
    pop ebp
    ret 4
```

Comentarii:

- ; definim un cadru de stiva cu EBP pivot/referinta
- ; alocam spatiu pe stiva pentru o variabila DWORD temporara
- ; salvam ebx pentru a-l putea restaura
- ; incarcam valoarea argumentului din stackframe
- ; testam coditia de terminare (n < 2)
- ; returnam 1 cand n < 2
- ; retinem valoarea lui n sa nu-l pierdem la apelul recursiv
- ; apelam factorial(n-1), am urcat in stiva valoarea n-1
- ; apel recursiv (STDCALL)
- ; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
- ; restauram valoarea lui n
- ; ebx <- factorial(n-1), preluat din variabila temporara
- ; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)
- ; refacem EBX la valoarea initiala
- ; eliberam memoria ocupata de variabila temporara
- ; restauram ebp la valoarea initiala
- ; STDCALL: revenire cu eliberare memorie parametri

(2) Variabile locale

Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de ieșire

- Exemplu: cod ieșire dintr-o funcție STDCALL asm (din apel recursiv) - stackframe

Diagram of a stack frame structure:

- ESP → Adresă Revenire
- Parametri
- Resurse volatile
- Registri nevolatili
- Variabile locale
- EBP → EBP apelant
- Adresă Revenire
- Parametri
- Resurse volatile
- date program apelant
- Adrese mari

```
factorial:
    push ebp
    mov ebp, esp
    sub esp, 4
    push ebx

    mov eax, [ebp + 8]
    cmp eax, 2
    jae .recursiv
    mov eax, 1
    jmp .gata

.recursiv:
    push eax
    dec eax
    push eax
    call factorial
    mov [ebp - 4], eax
    pop eax
    mov ebx, [ebp - 4]
    mul ebx

.gata:
    pop ebx
    add esp, 4
    pop ebp
    ret 4
```

Annotations for assembly code:

- `push ebp`: definim un cadru de stiva cu EBP pivot/referinta
- `mov ebp, esp`: alocam spatiu pe stiva pentru o variabila DWORD temporara
- `sub esp, 4`: salvam ebx pentru a-l putea restaura
- `push ebx`: incarcam valoarea argumentului din stackframe
- `mov eax, [ebp + 8]`: testam coditia de terminare ($n < 2$)
- `cmp eax, 2`: returnam 1 cand $n < 2$
- `jae .recursiv`: returnam 1 cand $n < 2$
- `mov eax, 1`: returnam 1 cand $n < 2$
- `jmp .gata`: returnam 1 cand $n < 2$
- `push eax`: retinem valoarea lui n sa nu-l pierdem la apelul recursiv
- `dec eax`: apelam factorial(n-1), am urcat in stiva valoarea n-1
- `push eax`: apel recursiv (STDCALL)
- `call factorial`: apel recursiv (STDCALL)
- `mov [ebp - 4], eax`: salvam in DWORD-ul temporar valoarea lui factorial(n-1)
- `pop eax`: restauram valoarea lui n
- `mov ebx, [ebp - 4]`: ebx ← factorial(n-1), preluat din variabila temporara
- `mul ebx`: calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)
- `pop ebx`: refacem EBX la valoarea initiala
- `add esp, 4`: eliberam memoria ocupata de variabila temporara
- `pop ebp`: restauram ebp la valoarea initiala
- `ret 4`: STDCALL: revenire cu eliberare memorie parametri

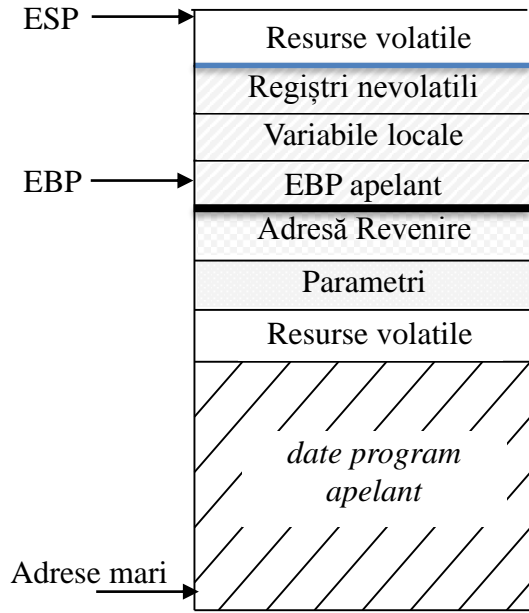
(3) Stackframe

Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de ieșire

- Exemplu: cod ieșire dintr-o funcție STDCALL asm (din apel recursiv) - stackframe

Diagrama structurii stackframe:



```
factorial:
    push ebp
    mov ebp, esp        ; definim un cadru de stiva cu EBP pivot/referinta
    sub esp, 4          ; alocam spatiu pe stiva pentru o variabila DWORD temporara
    push ebx            ; salvam ebx pentru a-l putea restaura

    mov eax, [ebp + 8]  ; incarcam valoarea argumentului din stackframe

    cmp eax, 2          ; testam coditia de terminare (n < 2)
    jae .recursiv
    mov eax, 1          ; returnam 1 cand n < 2
    jmp .gata

.recursiv:
    push eax            ; retinem valoarea lui n sa nu-l pierdem la apelul recursiv

    dec eax
    push eax            ; apeland factorial(n-1), am urcat in stiva valoarea n-1
    call factorial      ; apel recursiv (STDCALL)

    mov [ebp - 4], eax  ; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
    pop eax             ; restauram valoarea lui n
    mov ebx, [ebp - 4]  ; ebx <- factorial(n-1), preluat din variabila temporara
    mul ebx             ; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)

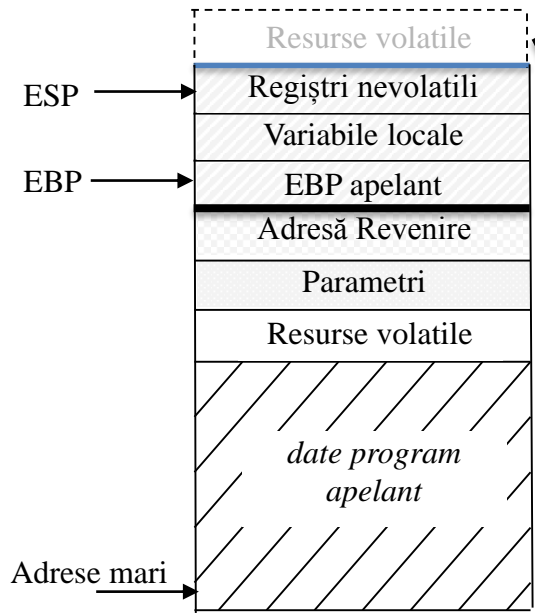
.gata:
    pop ebx             ; refacem EBX la valoarea initiala
    add esp, 4          ; eliberam memoria ocupata de variabila temporara
    pop ebp             ; restauram ebp la valoarea initiala
    ret 4               ; STDCALL: revenire cu eliberare memorie parametri
```

(4) Revenire și eliberare

Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de ieșire

- Exemplu: cod ieșire dintr-o funcție STDCALL asm (din apel inițial) - stackframe



ESP →

EBP →

Adrese mari →

```
factorial:
    push ebp
    mov ebp, esp        ; definim un cadru de stiva cu EBP pivot/referinta
    sub esp, 4          ; alocam spatiu pe stiva pentru o variabila DWORD temporara
    push ebx            ; salvam ebx pentru a-l putea restaura

    mov eax, [ebp + 8]  ; incarcam valoarea argumentului din stackframe

    cmp eax, 2          ; testam coditia de terminare (n < 2)
    jae .recursiv
    mov eax, 1          ; returnam 1 cand n < 2
    jmp .gata

.recursiv:
    push eax            ; retinem valoarea lui n sa nu-l pierdem la apelul recursiv

    dec eax
    push eax            ; apeland factorial(n-1), am urcat in stiva valoarea n-1
    call factorial      ; apel recursiv (STDCALL)

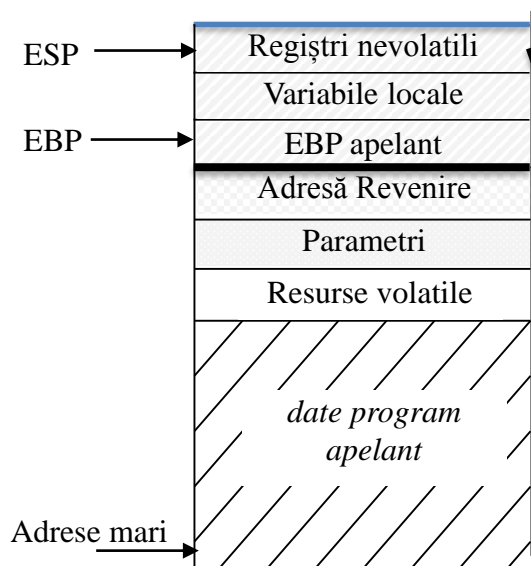
    mov [ebp - 4], eax   ; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
    pop eax             ; restauram valoarea lui n
    mov ebx, [ebp - 4]   ; ebx <- factorial(n-1), preluat din variabila temporara
    mul ebx             ; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)

.gata:
    pop ebx            ; refacem EBX la valoarea initiala
    add esp, 4          ; eliberam memoria ocupata de variabila temporara
    pop ebp            ; restauram ebp la valoarea initiala
    ret 4              ; STDCALL: revenire cu eliberare memorie parametri
```

Interfațarea cu limbajele de nivel înalt

- Apelul subrutinelor – cod de ieșire

- Exemplu: cod ieșire dintr-o funcție STDCALL asm (din apel inițial) - stackframe



```
factorial:
    push ebp
    mov ebp, esp        ; definim un cadru de stiva cu EBP pivot/referinta
    sub esp, 4          ; alocam spatiu pe stiva pentru o variabila DWORD temporara
    push ebx            ; salvam ebx pentru a-l putea restaura

    mov eax, [ebp + 8]   ; incarcam valoarea argumentului din stackframe

    cmp eax, 2           ; testam coditia de terminare (n < 2)
    jae .recursiv
    mov eax, 1           ; returnam 1 cand n < 2
    jmp .gata

.recursiv:
    push eax            ; retinem valoarea lui n sa nu-l pierdem la apelul recursiv

    dec eax
    push eax            ; apeland factorial(n-1), am urcat in stiva valoarea n-1
    call factorial      ; apel recursiv (STDCALL)

    mov [ebp - 4], eax   ; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
    pop eax             ; restauram valoarea lui n
    mov ebx, [ebp - 4]   ; ebx <- factorial(n-1), preluat din variabila temporara
    mul ebx             ; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)

.gata:
    pop ebx             ; refacem EBX la valoarea initiala
    add esp, 4          ; eliberam memoria ocupata de variabila temporara
    pop ebp            ; restauram ebp la valoarea initiala
    ret 4               ; STDCALL: revenire cu eliberare memorie parametri
```


Interfațarea cu limbajele de nivel înalt



- Apelul subrutinelor – cod de ieșire

- Exemplu: cod ieșire dintr-o funcție STDCALL asm (din apel inițial) - stackframe

```
factorial:
    push ebp
    mov ebp, esp           ; definim un cadru de stiva cu EBP pivot/referinta
    sub esp, 4             ; alocam spatiu pe stiva pentru o variabila DWORD temporara
    push ebx               ; salvam ebx pentru a-l putea restaura

    mov eax, [ebp + 8]     ; incarcam valoarea argumentului din stackframe

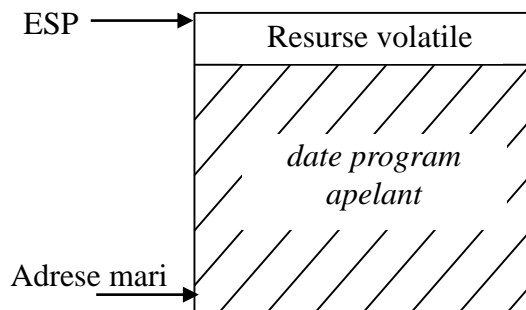
    cmp eax, 2             ; testam coditia de terminare (n < 2)
    jae .recursiv
    mov eax, 1             ; returnam 1 cand n < 2
    jmp .gata

.recursiv:
    push eax               ; retinem valoarea lui n sa nu-l pierdem la apelul recursiv

    dec eax
    push eax               ; apeland factorial(n-1), am urcat in stiva valoarea n-1
    call factorial          ; apel recursiv (STDCALL)

    mov [ebp - 4], eax     ; salvam in DWORD-ul temporar valoarea lui factorial(n-1)
    pop eax                ; restauram valoarea lui n
    mov ebx, [ebp - 4]     ; ebx <- factorial(n-1), preluat din variabila temporara
    mul ebx                ; calculam (edx:)eax = (EAX=)n * (EBX=)factorial(n-1)

.gata:
    pop ebx                ; refacem EBX la valoarea initiala
    add esp, 4             ; eliberam memoria ocupata de variabila temporara
    pop ebp                ; restauram ebp la valoarea initiala
    ret 4                  ; STDCALL: revenire cu eliberare memorie parametri
```





Bitdefender®