FII Iasi
- Arhitectura calculatoarelor si sisteme de operare
- Probabilități și Statistică

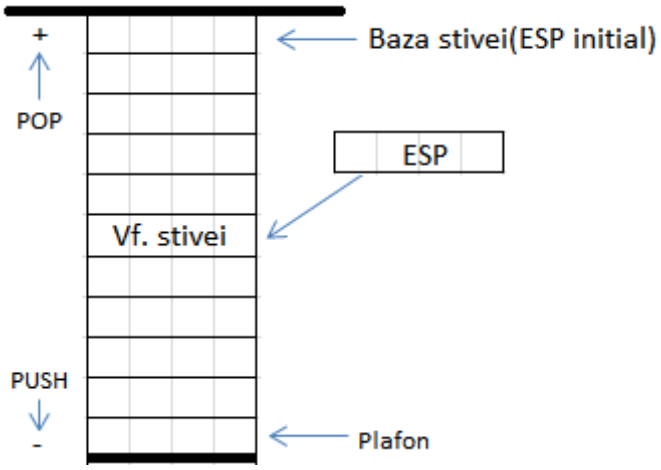
Orar

Sitemap

Laboratorul 10

* stiva (push, pop, pusha, popa, pushf, popf)
* Functii, apel - CALL, RET
* descrierea unui apel de funcție (parametri, variabile locale)

Stiva este o zona continua de memorie si poate avea maxim 4GB. Elementele se pot adauga si elimina de pe stiva folosind intructiunile **PUSH** si **POP**.
La adaugarea unui element pe stiva, procesorul decrementeaza valoarea registrului *ESP,* dupa care scrie valoarea (la noul varf al stivei).
La eliminarea unui element de pe stiva, procesorul citeste valoarea ce se afla in varful stivei, dupa care incrementeaza valoarea lui *ESP.*
Astfel putem spune ca *Stiva* creste "in jos" (adica elementele noi sunt adaugate la adrese mai mici de memorie), si se micsoreaza "in sus" (la eliminarea elementelor de pe stiva, aceasta se micsoreaza inspre adrese mai mari de memorie)



Adaugarea unui element pe stiva: **PUSH** *operand*
Eliminarea unui element de pe stiva: **POP** *operand*
Operand poate fi registru (16bit/32bit), zona de memorie sau valoare (immediate byte/word/dword value)

Instructiunile *PUSH* si *POP* decrementeaza si respectiv incrementeaza in mod automat valoarea lui *ESP* cu dimensiunea operandului. Astfel, instructiunea *PUSH AX* va decrementa *ESP* cu 2 unitati.
Daca instructiunea *PUSH* este folosita impreuna cu o valoare directa atunci, indiferent de dimensiunea sa, valoarea este adaugata pe stiva intr-o reprezentare pe 32 de biti, printr-o extindere fara semn.

Ex.

```
PUSH 033h
MOV EAX, dword ptr [ESP]
ADD ESP, 4
//pune in registrul EAX valoarea 0x00000033 ca un DWORD: reprezentare C2(32,0).
//Codul este echivalent cu MOV EAX,033h
```

Instructiunile *PUSH* si *POP* nu pot folosi registri sau zone de memorie de dimensiune 1 octet (Ex: *PUSH AL*).

Alinierea Stivei. Se recomanda ca stiva sa fie aliniata la 4 octeti (DWORD). De exemplu, daca doriti sa puneti pe stiva continutul registrului *AX*, folositi:

```
PUSH EAX
sau
SUB ESP,4
MOV word ptr [ESP], AX
```

Bineinteles, daca doriti sa extrageti de pe stiva valoarea pe care tocmai ati adaugat-o, trebuie sa aveti grija ca valoarea lui ESP sa fie incrementata cu aceeasi valoarea cu care a fost decrementata. Astfel puteti folosi:

```
POP EAX // valoarea cautata se afla acum in AX
 sau
MOV AX, word ptr [ESP]
ADD ESP,4
```

Pentru a pune pe stiva continutul tuturor registrilor generali se pot folosi instructiunile ***PUSHA/PUSHD.***
Instructiunea ***PUSHA*** pune pe stiva cei 8 registri generali pe 16 biti in aceasta ordine: AX, CX, DX, BX, SP, BP, SI, DI. Valoarea lui SP este cea initiala (inainte de executarea instructiunii). Dupa executarea acestei instructiuni, registrul ESP este mai mic cu 16 unitati.
Instructiunea ***PUSHAD*** pune pe stiva cei 8 registri generali pe 32 biti in aceasta ordine: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI. Valoarea lui ESP este cea initiala (inainte de executarea instructiunii). Dupa executarea acestei instructiuni, registrul ESP este mai mic cu 32 unitati.

Pentru a recupera de pe stiva registrii generali - in ordine inversa - se pot folosi instructiunile ***POPA/POPAD.*** Singurul registru care nu este recuperat este *ESP.* Acesta va fi incrementat cu 16, respectiv 32 unitati dupa executarea instructiunii.

Pentru a salva pe stiva/recupera continutul registrului EFLAGS se poate folosi instructiunea **PUSHFD/POPFD**

Apelul functiilor se realizeaza prin intermediul instructiunii **CALL**. Iesirea din functie se poate realiza prin intermediul instructiunii **RET.** In continuare sunt prezentate *CALL near* si *RET near* (procedura apelata/apelanta se afla in acelasi segment de cod cu procedura curenta)

La executia instructiunii **CALL**, procesorul se comporta in felul urmator:
1. Adauga pe stiva valoarea registrului *EIP.*
2. Incarca offsetul procedurii in registrul *EIP.*
3. Se executa prima instructiune din cadrul procedurii.

La executia instructiunii **RET**, procesorul se comporta in felul urmator:
1. Se face *POP* adresei de revenire de pe stiva in registrul *EIP.*

2. Daca instructiunea *RET* are un parametru (optional) specificat prin operandul *n*, se incrementeaza ESP cu valoarea *n*, pentru a elimina parametrii de pe stiva.
3. Se reia executia procedurii apelante.

Standardul MS VS C/C++ pentru apelul procedurilor este ***cdecl.*** In acest standard:
1. Stiva este curatata (adica parametrii sunt eliminati) de catre apelant.
2. Transmiterea argumentelor se face de la dreapta spre stanga.
3. Registrii EAX, ECX, EDX sunt salvati de catre procedura apelanta iar ceilalti de catre procedura apelata.

Transmiterea parametrilor. Parametrii pot fi transmisi prin intermediul registrilor de uz general, pe stiva, in ordine inversa.
Accesarea parametrilor actuali. Daca exista, primul parametru se gaseste la adresa *EBP+8.* Datorita alinierii stivei la 4 octeti, urmatorii parametri se gasesc pe stiva, la adresele *EBP+12, EBP+16,* etc (din 4 in 4 octeti).
Returnarea valorilor. Valorile numerice intregi (int, short, etc) si adresele sunt returnate prin intermediul registrului *EAX*.

Ex

```
int suma(int a, int b, int c){
   return a+b+c;
}

void main(){
   int s;
   s = suma( 10,20,30 );
}

//-------Echivalent ASM-------------------------

int suma(int,int,int){
   _asm{
      mov eax, [ebp+8] //primul parametru
      add eax, [ebp+12]// al II-lea parametru
      add eax, [ebp+16]// al III-lea parametru
      //se returneaza prin intermediul registrului EAX
   }
}
void main(){
   _asm{
```

```
      //se pun pe stiva parametrii in ordine inversa
      push 30
      push 20
      push 10
      call suma //apelul procedurii
      add esp, 12 //eliminarea parametrilor de pe stiva
   }
}
```

Cadrul de stiva (Stack Frame) reprezinta o zona de memorie de pe stiva, asociata unei proceduri, care contine spatiul necesar salvarii parametrilor actuali, a variabilelor locale, a contextului de apel (adresa de revenire) si alte variabile temporale. Cand procedura apelata se termina de executat, cadrul de stiva asociat este eliminat iar executia threadului curent este reluata cu prin continuarea procedurii apelante.

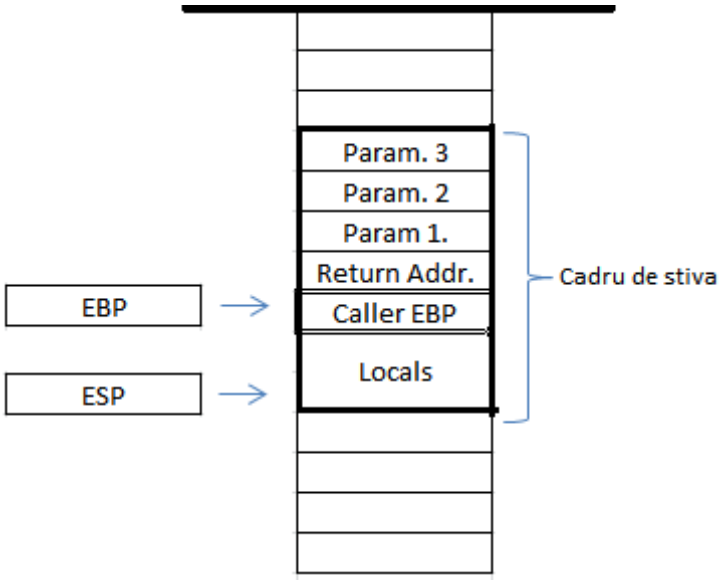
In cazul multor compilatoare (inclusiv MS VS) partea de inceput a functiilor este una standard:

```
PUSH EBP       //se salveaza base pointer pentru functia apelanta
MOV EBP, ESP //EBP se suprascrie cu adresa varfului curent al stivei
SUB ESP, X     //se face loc pentru variabilele locale
```

La fel si partea de sfarsit:

```
MOV ESP, EBP
POP EBP
RET
```

Astfel, in timpul executiei unei functii, stiva arata astfel:



Ex 1

Sa se scrie o functie f(int n) ce primeste ca parametru variabila int n si returneaza rezultatul calculului:
2 + 2*4 + 2*4*6 + 2*4*6*8 + … +2*4*6*8*…*2n.
Functia trebuie sa fie apelata din limbaj de asamblare.

Ex 2

```
#include <stdio.h>

//Calculul factorialului unui numar - recursiv

unsigned int fact_rec(unsigned int nr){
 _asm{
 //completati
 }
}


void main(){
 int nr,fact;
```

```
 printf("nr = ");
 scanf("%u",&nr);
 _asm{
 //completati
 }
 printf("%u! = %u",nr,fact);
}
```

,

l10p1.txt (0k)	Alexandru Baetu, 8 ian. 2015, 00:03	v.2	
l10p2.txt (1k)	Alexandru Baetu, 8 ian. 2015, 00:09	v.2	