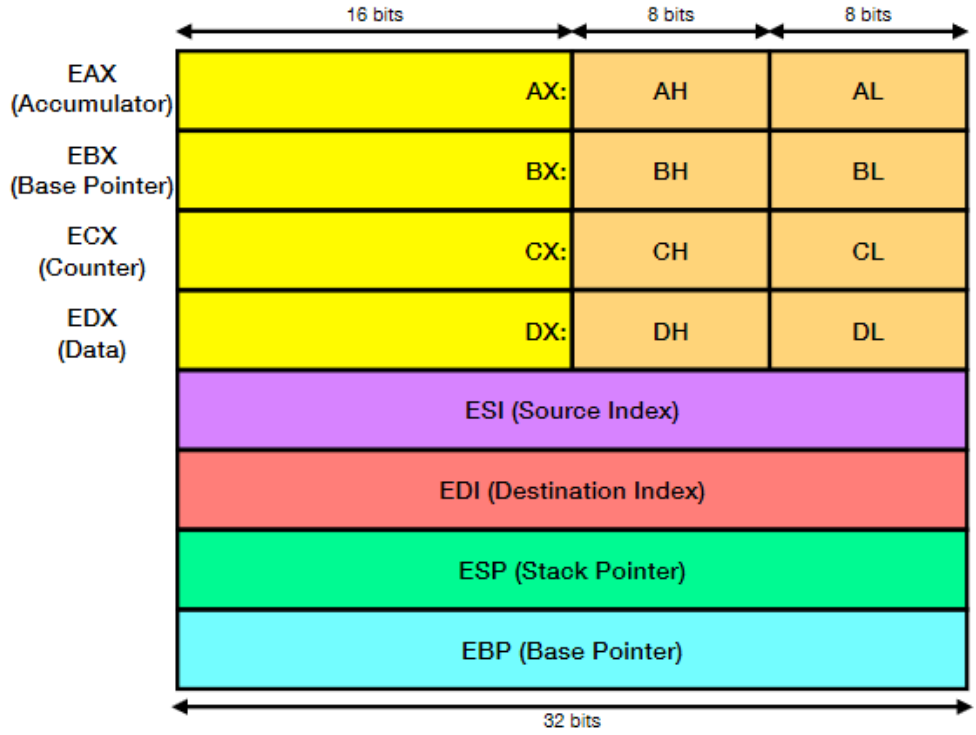


Registrii EFLAGS (indicatori) sunt:

- **CF** - *carry flag*, care se seteaza daca dupa ultima operatie a avut loc transport;
- **PF** - *parity flag*, care este setat daca numarul de biti de 1 din rezultatul ultimei operatii este par;
- **ZF** - *zero flag*, care este setat daca rezultatul ultimei operatii a fost 0;
- **SF** - *sign flag*, care este setat daca rezultatul ultimei operatii a fost negativ;
- **OF** - *overlow flag*, care este setat daca rezultatul ultimei operatii a produs overflow.

1. **byte** - dupa cum ii spune si numele, ocupa 1 byte, adica 8 biti in memorie;
2. **single** - ocupa 4 bytes (32 de biti) si este utilizat pentru stocarea numerelor fractionare;
3. **word** - ocupa 2 bytes (16 biti) si este utilizat pentru stocarea intregilor;
4. **long** - ocupa 4 bytes (32 de biti) si este utilizat pentru a stoca intregi pe 32 de biti sau numere fractionare pe 64 de biti. **Double word** (dword) ocupa tot 32 de biti;
5. **quad** - ocupa 8 bytes (64 de biti);
6. **ascii** - este utilizat pentru declararea sirurilor de caractere care nu sunt finalizate cu terminatorul de sir - nerecomandat;
7. **asciz** - este utilizat pentru declararea sirurilor de caractere care sunt finalizate cu terminatorul de sir;
8. **space** - defineste un spatiu in memorie, a carui dimensiune o specificam, de exemplu **.space 4**, insemnand ca se lasa 4 bytes = 32 de biti. Este util atunci cand declaram variabile pe care le calculam in cadrul programului si pentru care nu vrem initial o valoare default.

Instructiune	Efect
add op1, op2	op2 := op2 + op1
sub op1, op2	op2 := op2 - op1
mul op	(edx, eax) := eax × op
imul op	(edx, eax) := eax × op
div op	(edx, eax) := (edx, eax) / op
idiv op	(edx, eax) := (edx, eax) / op

Instructiune	Efect
not op	op := ∼op
and op1, op2	op2 := op2 & op1
or op1, op2	op2 := op2 | op1
xor op1, op2	op2 := op2 ^ op1

Instructiune	Efect
shr numar, op	op := op ≫ numar
shl numar, op	op := op ≪ numar
sar numar, op	op2 := op ≫ numar (cu pastrare semn op)
sal numar, op	op2 := op ≪ numar (cu pastrare semn op)

Operator	Descriere
jc	jump daca este carry setat
jnc	jump daca nu este carry setat
jo	jump daca este overflow setat
jno	jump daca nu este overlow setat
jz	jump daca este zero setat
jnz	jump daca nu este zero setat
js	jump daca este sign setat
jns	jump daca nu este sign setat
Operatori pentru numere fara semn	
jb	jump if below (op1 < op2)
jbe	jump if below or equal (op1 <= op2)
ja	jump if above (op1 > op2)
jae	jump if above or equal (op1 >= op2)
Operatori pentru numere cu semn	
jl	jump if less than (op1 < op2)
jle	jump if less than or equal (op1 <= op2)
jg	jump if greater than (op1 > op2)
jge	jump if greater than or equal (op1 >= op2)
Operatori de egaliate	
je	jump if equal (op1 == op2)
jne	jump if not equal (op1 ! = op2)

Observatii:

- je <=> cu jz (pentru ca cmp pune in tmp = op1 - op2 caz in care daca sunt egale da 0 adica ZF= 1 adica jz da true)
- a(b ,c ,d)
  a – adresa de memore/constanta
  b – registru
  c – registru (%ecx)
  d – constanta

  v[2] = 5

  caz a = 0

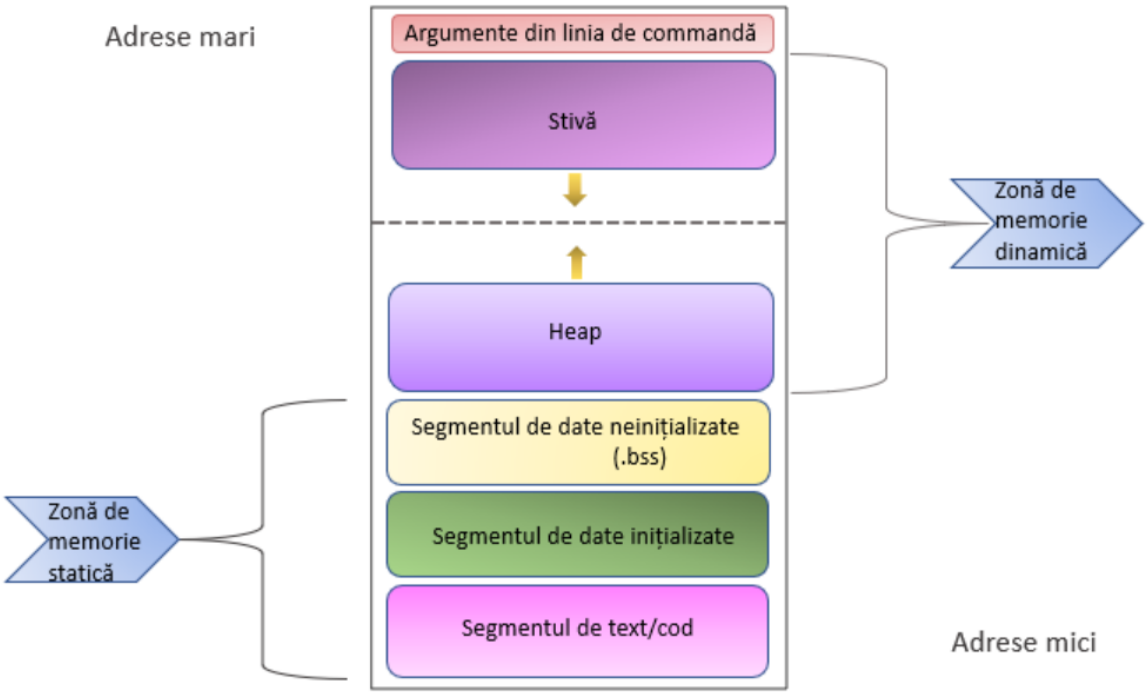
  ```
  mov $v, %edi
  mov $2, %ecx
  movl $5, (%edi, %ecx, 4)
  ```

  caz b = 0

  ```
  movl $5, v( ,%ecx,4)
  ```

1 Stiva

Stiva este o regiune dinamică în cadrul unui proces care funtioneaza pe principul LIFO (Last In – First Out). La fiecare apel de funcție, este alocată o zonă de memorie pe stivă în care vor fi stocate **argumentele funcției, variabilele locale** și **adresa de retur**. La fiecare alocare stiva va crește "în jos", spre adrese mici, așa cum este prezentat și în figura 1.

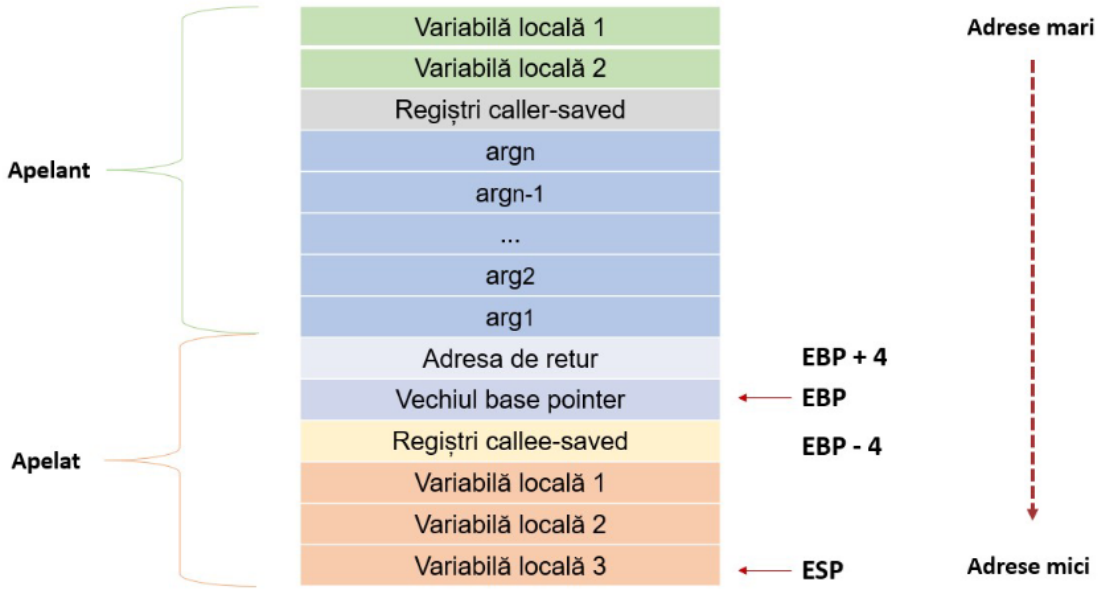


Stack pointer-ul %esp este registrul care indică întotdeauna spre vârful stivei și prin care se efectueaza, de fapt, push-urile si pop-urile. De exemplu, la fiecare **pushl** efectuat, %esp scade cu 4 (stiva crește spre adrese mici) si este salvata o valoare relativ la adresa de memorie indicata de %esp. Pentru fiecare **popl** efectuat, valoarea din varful stivei este depozitata in operandul instructiunii, iar %esp creste cu 4.

Urmarind scrierea studiata in laboratorul trecut a(b, c, d) , valoarea efectiva din varful stivei este data de **0(%esp)**. In plus, trebuie remarcat ca varful stivei va indica intotdeauna spre ultimul byte al ultimului element adăugat pe stiva.

Base pointer-ul este registrul care indică întotdeauna spre baza stivei și prin care se va face delimitarea cadrelor de apel și accesarea elementelor corespunzătoare acestora.

- **callee-saved : %ebx, %esp, %ebp, %esi, %edi**. Valorile acestor registri se garanteaza a fi restaurate de catre procedura, adica acestea trebuie salvate in zona de variabile locale de catre procedura.
- **caller-saved : %eax, %ecx, %edx**. Nu este garantata restaurarea lor. Apelantul trebuie sa salveze aceste valori inainte de incarcarea argumentelor functiei daca doreste sa regaseasca valorile initiale la iesirea din functie.



- `%d` - pentru afisarea int-urilor;
- `%ld` - pentru afisarea long-urilor;
- `%hu` - pentru afisarea short int-urilor (word-urilor);
- `%s` - pentru afisarea sirurilor de caractere.

Observatii:

- pot fi adaugate pe stiva doar word-uri si long-uri
- pusha (pune toate valorile registrilor pe stiva, si la descarcare pastreaza valorile)
- Cand stiva creste esp ul scade, ebp ramane la aceeasi valoare
- 0(%esp) va fi mereu adresa urmatoarei instructiuni, deci vom pleca de la 4(%esp)
- In cazul in care se doreste returnare, valorile vor fi depozitate in aceasta ordine in functie de numarul lor in %eax, %ecx, %edx si ulterior prin varful stivei