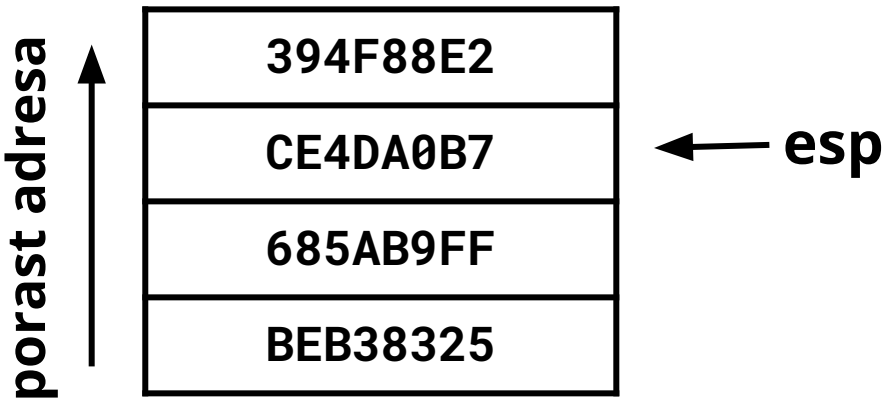
Arhitektura računara

Potprogrami

Stek x86 procesora

- LIFO struktura
- Registri:
  - **esp**
  - **ebp**
- Naredbe:
  - **push**
  - **pop**

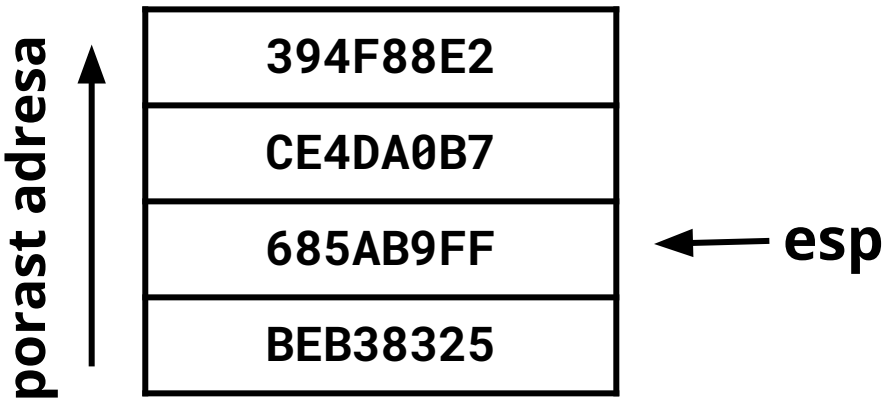
- Stek tokom **pushl $0**



Stek x86 procesora

- LIFO struktura
- Registri:
  - **esp**
  - **ebp**
- Naredbe:
  - **push**
  - **pop**

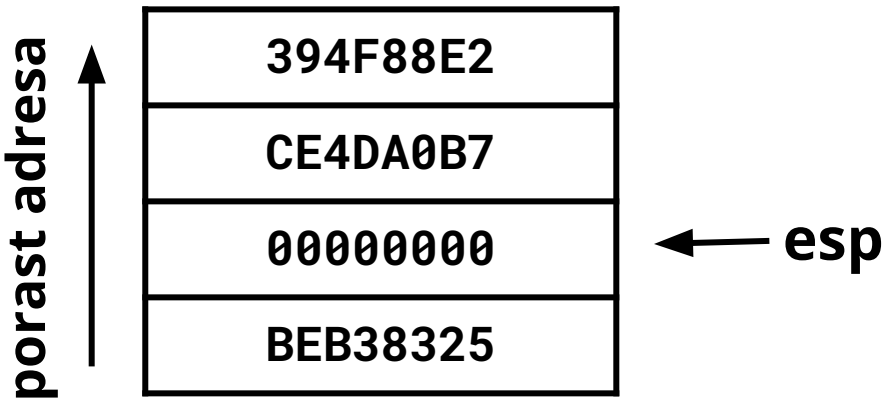
- Stek tokom **pushl $0**



Stek x86 procesora

- LIFO struktura
- Registri:
  - **esp**
  - **ebp**
- Naredbe:
  - **push**
  - **pop**

- Stek tokom **pushl $0**



Poziv potprograma i stek

- **Poziv**: naredba **call**
  - na stek ide adresa naredne naredbe
  - skače se na labelu-potprogram
- **Povratak**: naredba **ret**
  - sa steka uzima adresu povratka
  - skače na nju

Konvencija poziva potprograma

- Način poziva potprograma, prenosa parametara i rada sa lokalnim promenljivim
- Programski jezik C na x86 arhitekturi koristi **cdecl konvenciju**:
  - kod koji poziva je odgovoran za alociranje/dealociranje parametara
  - Parametri se smeštaju od poslednjeg ka prvom (*right-to-left* redosled)
  - povratna vrednost u registru **eax** (32-bitna), odnosno u **edx:eax** (64-bitna)
  - registri slobodni za korišćenje (*scratch* registri) su **eax**, **ecx** i **edx**
    - vrednosti ostalih registara se moraju restaurirati pre povratka
  - lokalne promenljive se nalaze na steku

Postupak tokom cdecl poziva

a) Stavi parametre na stek (redosled!)
b) Pozovi funkciju
c) Sačuvaj i podesi ebp
d) Alociraj lokalne promenljive na steku
e) Sačuvaj registre koji se koriste u funkciji

f) Izvrši kod funkcije

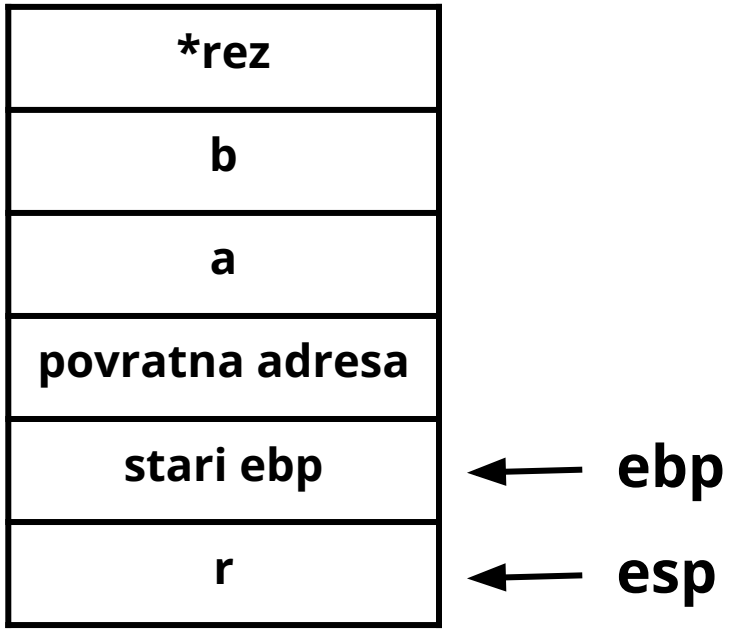
k) Skloni parametre sa steka
j) Vrati se iz funkcije
i) Restauriraj ebp
h) Oslobodi lokalne promenljive
g) Restauriraj sačuvane registre (redosled!)

Stek frejm

- Struktura koja se formira na steku tokom cdecl poziva
- Elementi:
  - **argumenti**
  - **povratna adresa**
  - **zatečena vrednost pokazivača frejma**
  - **lokalne promenljive**
- **Pokazivač frejma** - registar **ebp**

Stek frejm

```
void nzd (int a, int b, int *rez) {
    int r;
    while (a != b) {
        if (a > b)
            a -= b;
        else
            b -= a;
    }
    r = a;
    *rez = r;
}
```



Potprogrami na asembleru

- Definisanje: **labelom**
- Primer:

```
nzd:
    pushl %ebp
    …
```

- Dibager:
  - **step** - potprogram korak po korak
  - **next** - izvršavanje u celini

Potprogrami na assembler (cdecl)

- **Poziv potprograma**

```
pushl %ecx          # treći argument -> stek
pushl %ebx          # drugi argument -> stek
pushl %eax          # prvi argument -> stek
call nzd
addl $12, %esp      # oslobađanje prostora
```

- **Početak potprograma**

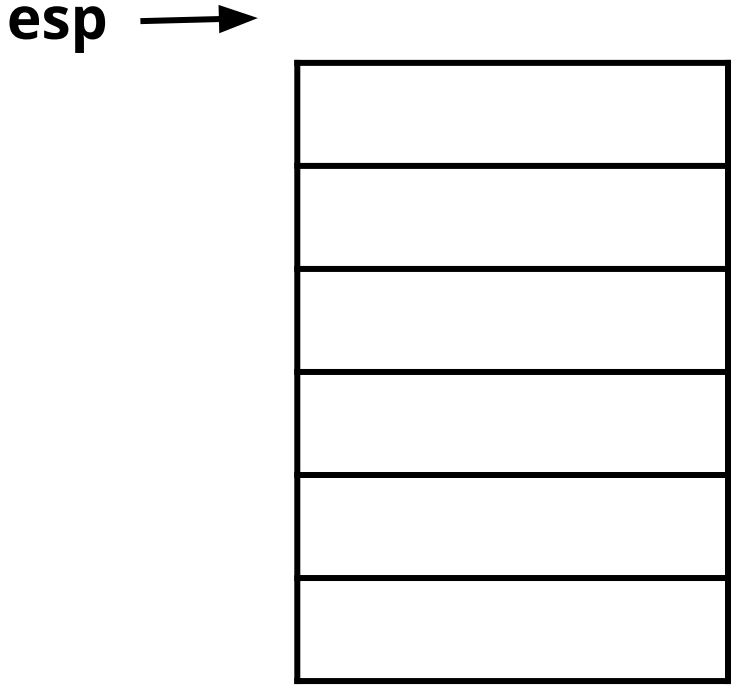
```
pushl %ebp          # čuvanje zatečenog FP-a
movl %esp, %ebp     # postavljanje novog FP-a
subl $4, %esp       # prostor za lokalnu promenljivu
```

- **Kraj potprograma**

```
movl %ebp, %esp     # brisanje lokalne promenljive
popl %ebp           # vraćanje prethodnog FP-a
ret
```

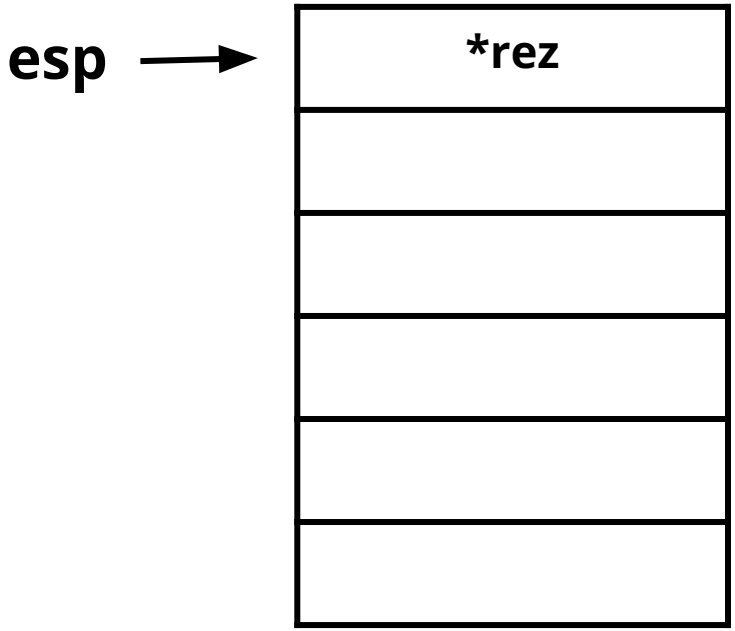
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd
    addl $12, %esp
    …

nzd:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    …
    movl %ebp, %esp
    popl %ebp
    ret
```



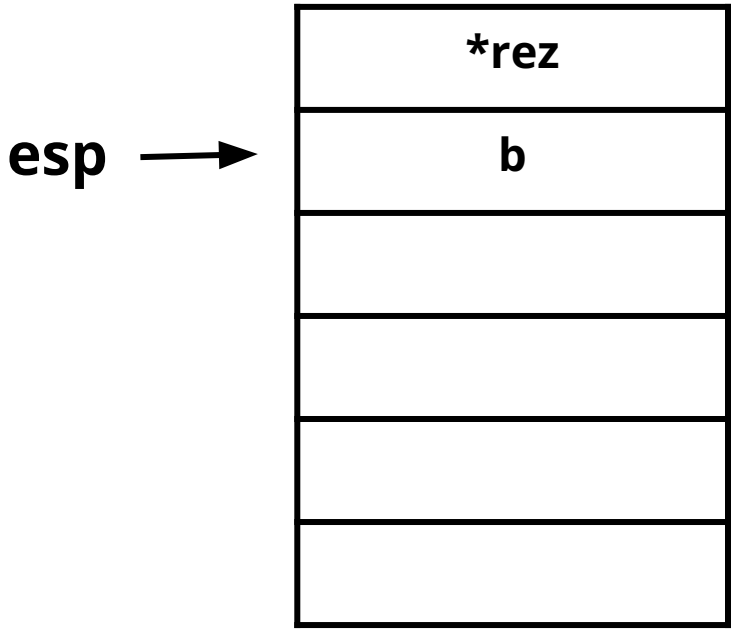
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
```



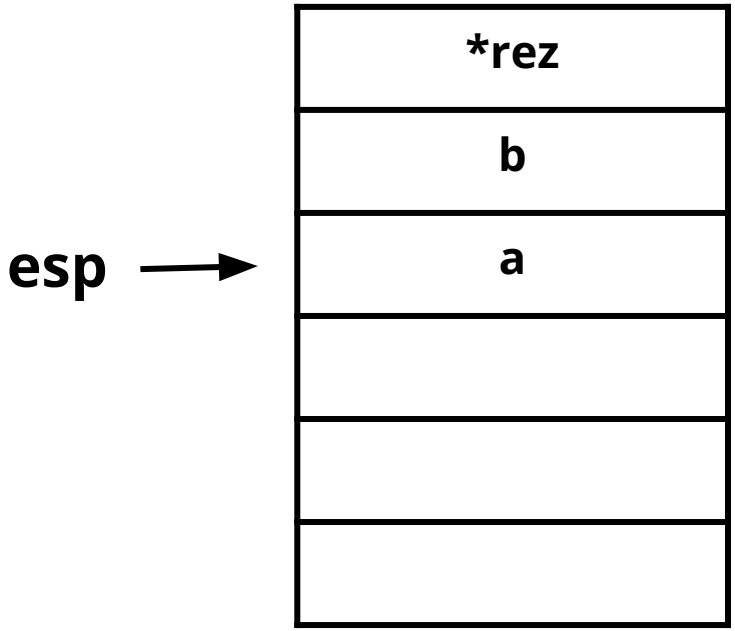
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
```



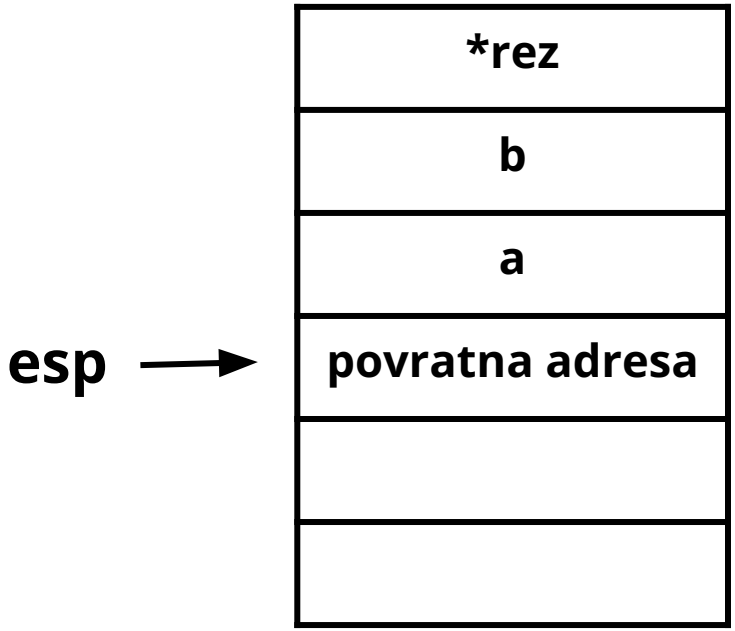
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
```



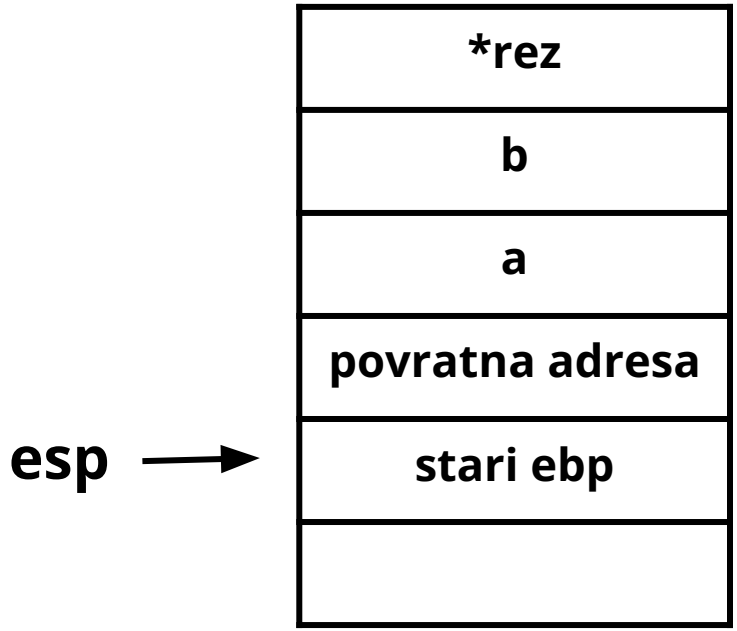
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd
```



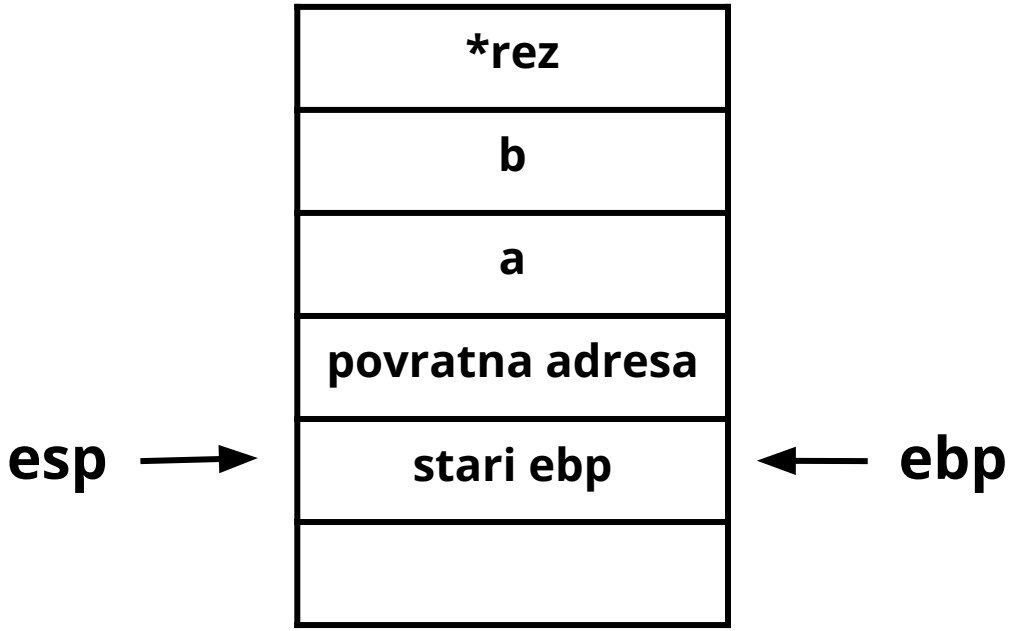
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd


nzd:
    pushl %ebp
```



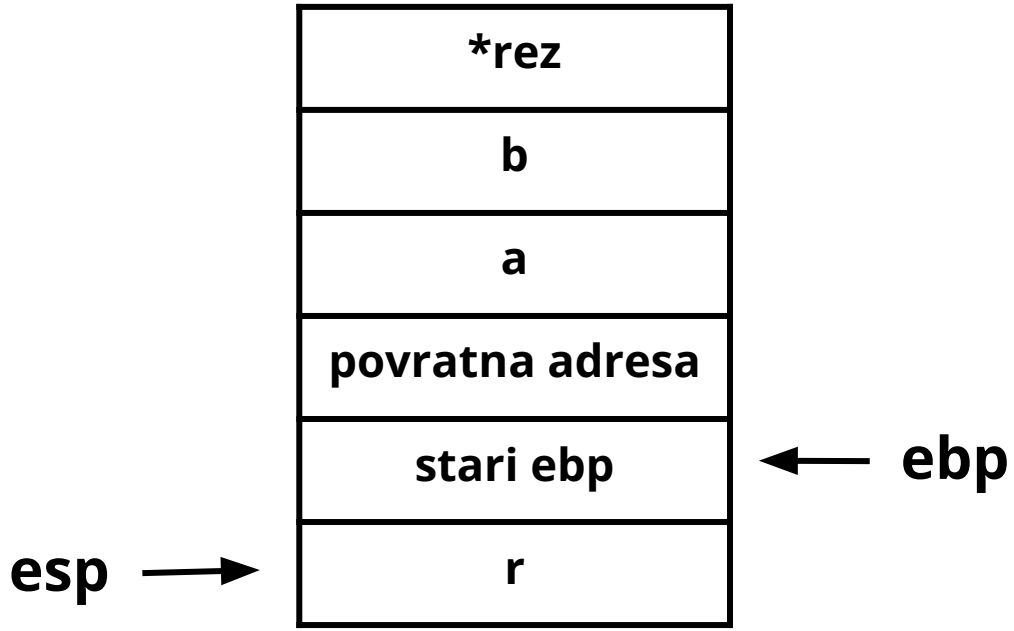
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd


nzd:
    pushl %ebp
    movl %esp, %ebp
```



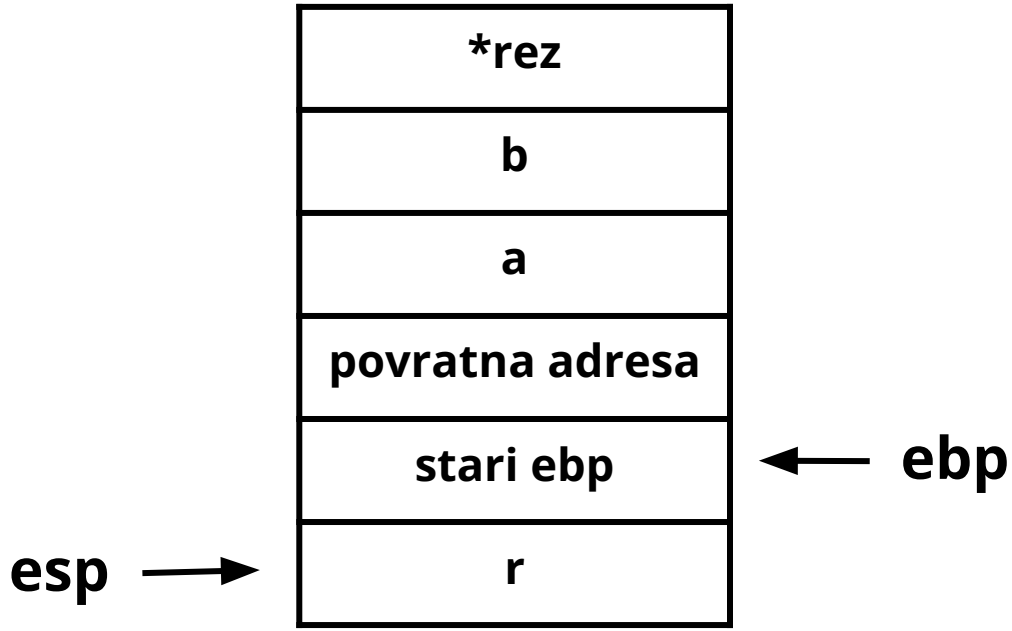
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd


nzd:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
```



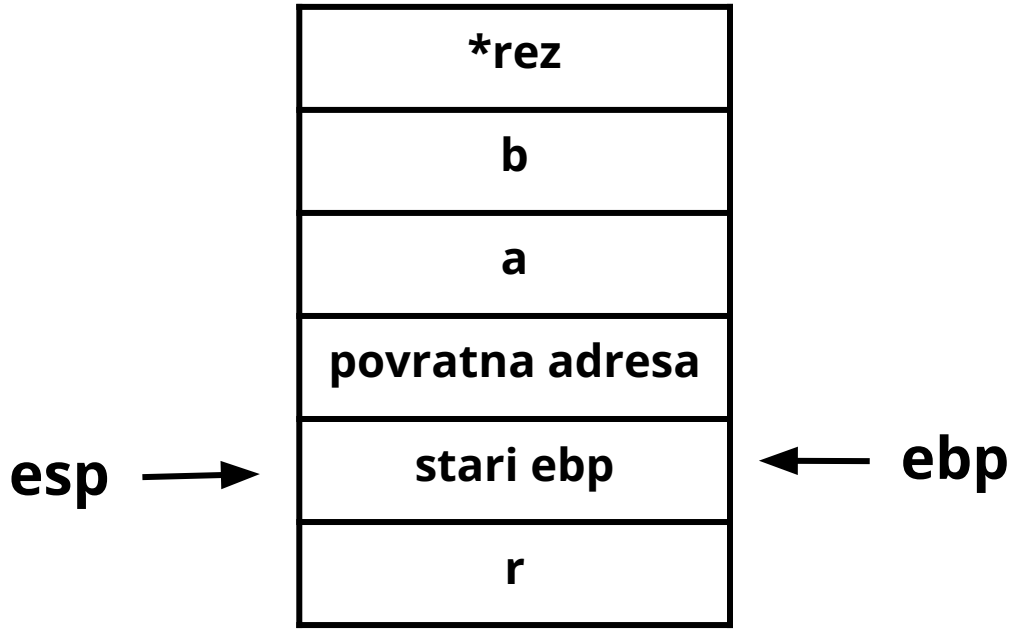
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd


nzd:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    ...
```



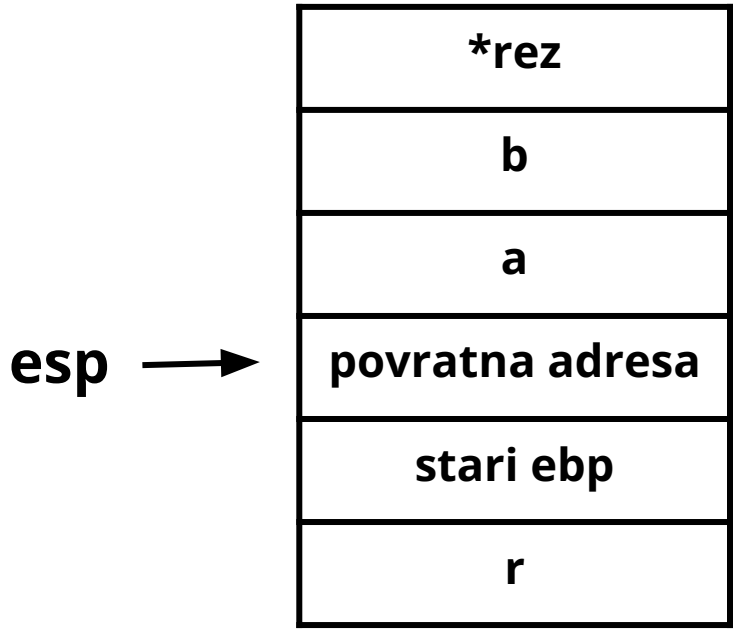
Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd


nzd:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    ...
    movl %ebp, %esp
```



Potprogrami na asembleru (cdecl)

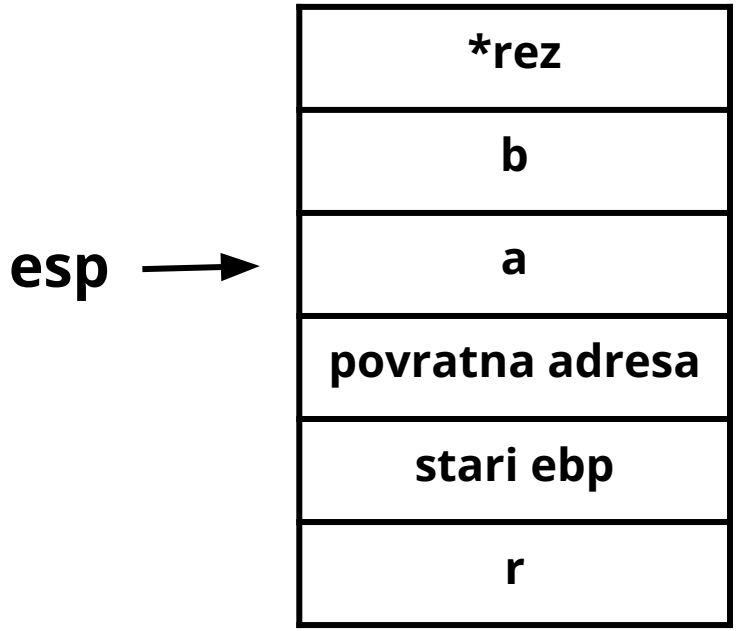
```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd


nzd:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    ...
    movl %ebp, %esp
    popl %ebp
```



ebp = stari ebp

Potprogrami na asembleru (cdecl)

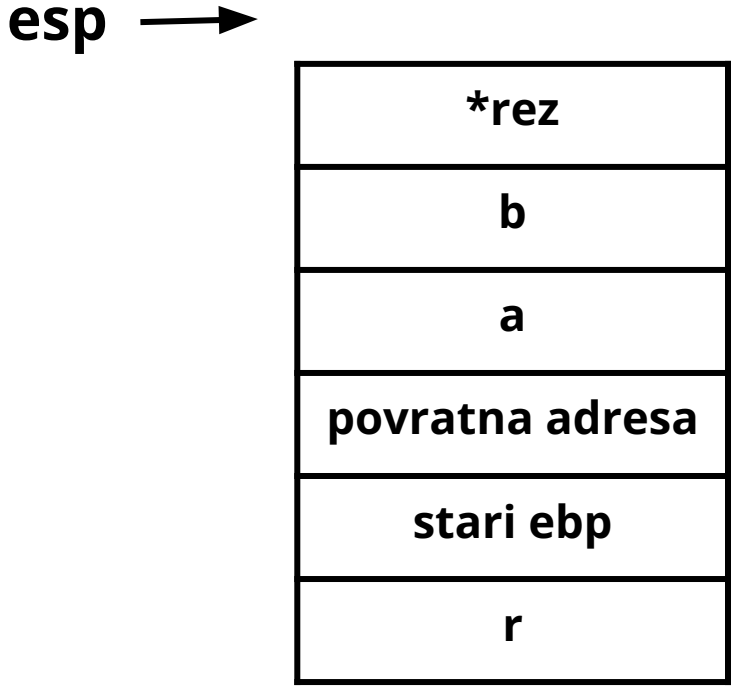
```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd


nzd:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    ...
    movl %ebp, %esp
    popl %ebp
    ret
```



eip = povratna adresa

Potprogrami na asembleru (cdecl)

```
main:
    …
    pushl %ecx
    pushl %ebx
    pushl %eax
    call nzd
    addl $12, %esp

nzd:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    ...
    movl %ebp, %esp
    popl %ebp
    ret
```



Pristup parametrima i lokalnim promenljivim

...

12(%ebp)	Drugi parametar funkcije
8(%ebp)	Prvi parametar funkcije
4(%ebp)	Povratna adresa (stari eip)
(%ebp)	Stari ebp
-4(%ebp)	Prva lokalna promenljiva
-8(%ebp)	Druga lokalna promenljiva

...

```
movl 12(%ebp), %ebx      # drugi parametar (b) → ebx
movl -4(%ebp), %ecx      # prva lokalna promenljiva (r) → ecx
```

C deklaracija funkcije

```
void saberi(int a, int b, int *rez);
```

- Potprogram treba da se zove **saberi**
- Potprogram **ne vraća vrednost** (znači da je sadržaj registra eax nebitan po izlasku iz potprograma)
- Potprogram ima **3 parametra**, prva dva se prenose po vrednosti, dok se treći prenosi po adresi
- Nakon inicijalizacije stek frejma, **prvi parametar** će se nalaziti na **ebp+8**, drugi na **ebp+12**, a treći na **ebp+16**
- Da bi se pristupilo vrednosti parametra prenetog po adresi, treba koristiti neku od varijanti indirektnog adresiranja

Prenos argumenata po vrednosti i po adresi

- **Po vrednosti:**
  - na stek se stavlja kopija vrednosti argumenta
  - njegova izmena u potprogramu ne menja izvorni argument
- **Po adresi:**
  - na stek se stavlja adresa argumenta
  - pošto se za pristup koristi varijanta indirektnog adresiranja, izmena vrednosti menja izvorni argument

Prenos argumenata po vrednosti i po adresi

- Primer

```
# void nzd (int a, int b, int *rez)

# treći parametar (adresa rezulatata) -> esi
movl 16(%ebp), %esi
movl $0, (%esi)     # postavljanje rezultata na 0
```

Primer: Potprogram za sabiranje brojeva

```
# int saberi (int a, int b);
a: .long 123
b: .long 456

saberi:                     # naziv potprograma
     pushl %ebp             # početak potprograma
     movl %esp, %ebp

     movl 8(%ebp), %eax     # telo potprograma
     addl 12(%ebp), %eax

     movl %ebp, %esp        # završetak potprograma
     popl %ebp
     ret

main:                       # glavni program
     pushl b                # drugi argument -> stek
     pushl a                # prvi argument -> stek
     call saberi            # poziv potprograma
     addl $8, %esp          # oslobađanje prostora na steku

kraj:
     movl $1, %eax
     movl $0, %ebx
     int $0x80
```

Linkovanje više fajlova u jedan program

- U potprogramu (potprogram bi trebalo da koristi samo stek frejm):
  - **.globl ime_potprograma**
- U glavnom programu:
  - **.globl main**
- Kompajliranje + linkovanje:
  - **gcc -m32 -g -o program pp.S glavni.s**

Linkovanje više fajlova u jedan program

- **gcc -m32 -g -o saberi saberipp.S saberigl.S**
- saberipp.S:

```
# potprogram za sabiranje dva broja
# int saberi(int a, int b);

.section .text
.globl saberi
saberi:                         # naziv potprograma
   pushl %ebp                   # početak potprograma
   movl %esp, %ebp

   movl 8(%ebp), %eax           # telo potprograma
   addl 12(%ebp), %eax

   movl %ebp, %esp              # završetak potprograma
   popl %ebp
   ret
```

Linkovanje više fajlova u jedan program

- saberigl.S:

```
# glavni program za sabiranje dva broja
# int saberi(int a, int b);

.section .data
    a: .long 123
    b: .long 456
.section .text
.globl main
main:               # glavni program
    pushl b         # drugi argument -> stek
    pushl a         # prvi argument -> stek
    call saberi     # poziv potprograma
    addl $8, %esp   # oslobađanje prostora na steku

kraj:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

Linkovanje više fajlova u jedan program

- **gcc -m32 -g -o saberi saberipp.S saberigl.c**
- saberigl.c:

```
// glavni program za sabiranje dva broja
#include <stdio.h>

int saberi(int a, int b); // deklaracija asemblerskog potprograma

int main() {
    int rez;
    rez = saberi(8, 12);
    printf("Rezultat za 8 i 12 je %d\n", rez);
}
```