

ASEMBLER – stek i procedure

Primer 1: Računanje faktorijela broja koji se unosi sa tastature

```
; Stampati faktorijel broja koji se unosi sa tastature
#include <stdio.h>
;
;int main()
;{
;    int broj, faktorijel=1, i;
;
;    printf("Unesite broj: ");
;    scanf("%d", &broj);
;
;    for (i=broj; i>0; i--)
;        faktorijel *= i;
;
;    printf("Faktorijel broja %d je %d.\n", broj, faktorijel);
;
;    return 0;
;};
;

extern printf, scanf

segment .data
    poruka1      db      "Unesite broj: ", 0
    poruka2      db      "Faktorijel broja %u je %u", 10, 0
    fmt          db      "%u",0
    faktorijel   dd      1

segment .bss
    broj         resd    1          ; broj ciji se faktorijel trazi

segment .text
    global main
main:
    enter    0,0                    ; rutina za inicijalizaciju

    ; Ovde pocinje koristan kod
    push    poruka1
    call    printf                  ; printf("Unesite broj: ");
    add     esp, 4

    push    broj
    push    fmt
    call    scanf                  ; scanf("%u", &broj );
    add     esp, 8

    mov     ecx, [broj]

petlja:
    mov     eax, [faktorijel]       ; loop petlja
    mul     ecx                    ; eax *= ecx
    mov     [faktorijel], eax      ; faktorijel = edx:eax (visih 32 bita koji se nalaze u edx se
zanemaruju)
    loop    petlja                 ; skok na pocetak petlje

    push    dword [faktorijel]
    push    dword [broj]
    push    poruka2
    call    printf
    add     esp, 12

    mov     eax, 0                  ; vrati se nazad u OS
    leave
    ret
```


Primer 2:

```

; Pronaci zbir cifara broja koji se unosi sa tastature
;
#include <stdio.h>
;
int main()
{
    int broj, cifra, kolicnik, suma=0;

    printf("Unesite broj: ");
    scanf("%d", &broj);

    kolicnik=broj;
    while (kolicnik>0)
    {
        cifra = kolicnik%10;
        suma += cifra;
        kolicnik /= 10;
    }

    printf("Suma cifara broja %d je %d\n", broj, suma);

    return 0;
}

extern printf, scanf
segment .data
    poruka1      db      "Unesite broj: ", 0
    poruka2      db      "Zbir cifara broja %u je %u", 10, 0
    fmt          db      "%u",0

segment .bss
    broj         resd     1          ; broj ciji se zbir cifara trazi

segment .text
global main
main:
    enter        0,0                ; rutina za inicijalizaciju

    ; Ovde pocinje koristan kod
    push        poruka1
    call        printf
    add         esp, 4

    push        broj
    push        fmt
    call        scanf                ; scanf("%u", &broj);
    add         esp, 8

    mov         ecx, 0                ; u ecx ce se nalaziti suma cifara, inicijalizacija na nulu
    mov         eax, [broj]

while_petlja:
    ; while ( kolicnik > 0 )
    cmp         eax, 0
    jna         end_while_petlja     ; izlazi iz petlje ako je !(kolicnik > 0)

    ; edx:eax / 10 = eax sa ostatkom u edx
    mov         edx, 0                ; setuje se edx=0 jer se deljenik vec nalazi u eax
    mov         ebx, 10               ; u ebx registar se unese delilac 10
    div         ebx                   ; deljenje edx:eax / ebx = eax sa ostatkom edx

    add         ecx, edx              ; ostatak, tj. cifra u edx registru i dodaje se postojećoj sumi u ecx
    jmp         while_petlja          ; skok na pocetak while petlje
end_while_petlja:

    push        ecx
    push        dword [broj]
    push        poruka2
    call        printf
    add         esp, 12

    mov         eax, 0                ; vrati se nazad u C
    leave
    ret

```

1. Stek

Mnogi procesori imaju ugrađenu podršku za stek. Stek je struktura liste tipa *Last In First Out* (LIFO). Stek je prostor u memoriji koji je organizovan na ovaj način. PUSH instrukcija dodaje podatke na stek, dok POP instrukcija sklanja podatke sa steka. Podatak koji se sa steka skida je uvek onaj poslednji koji je na njega postavljen.

SS segmentni registar specificira segment u koji je stek smešten (a to je obično segment u kome se nalaze podaci). ESP registar sadrži adresu prvog podatka koji se može skinuti sa steka. Za ovakav podatak se kaže da stoji na *vrhu steka*.

PUSH instrukcija dodaje *double word* podatak (4 bajta) na stek, oduzimajući 4 od ESP i postavljajući dati podatak na [ESP]. POP instrukcija čita *double word* podatak na [ESP], a zatim dodaje 4 na ESP. Kod koji sledi demonstrira kako se koriste ove instrukcije, pod pretpostavkom da je ESP inicijalno setovano na adresu 1000h:

```
push dword 1          ; 1 je sacuvano na 0FFCh, ESP = 0FFCh
push dword 2          ; 2 je sacuvano na 0FF8h, ESP = 0FF8h
push dword 3          ; 3 je sacuvano na 0FF4h, ESP = 0FF4h
pop  eax              ; EAX=3, ESP=0FF8h
pop  ebx              ; EAX=2, ESP=0FFCh
pop  ecx              ; EAX=1, ESP=1000h
```

Stek se može koristiti za privremeno čuvanje podataka. Takođe se koristi za pozive potprogramima, prosleđivanje parametara potprogramima itd.

80x86 procesori imaju i PUSHA instrukciju koja na stek stavlja vrednosti registara EAX, EBX, ECX, EDX, ESI, EDI i EBP. Instrukcija POPA vraća te vrednosti natrag u navedene registre. Ove instrukcije se koriste za pamćenje stanja u odrenim tačkama u programu.

2. CALL i RET instrukcije

80x86 procesori daju dve instrukcije koje korišćenjem steka čine poziv potprograma jednostavnim i brzim. CALL instrukcija pravi bezuslovni skok na potprogram i na stek stavlja adresu sledeće instrukcije (*push*). RET instrukcija skida (*pop*) adresu sa steka i nastavlja izvršavanje od te adrese. Kada se koriste ove instrukcije, veoma je važno voditi računa da li se prave vrednosti stavljaju na stek i skidaju sa njega.

Sledi primer čiji je zadatak da učitava broj sa tastature, a zatim odštampa zbir cifara svih brojeva od 1 do zadatog broja. Zbir cifara se računa u potprogramu. Broj čiji se zbir cifara računa se smešta u registar EAX, dok potprogram vraća rezultat (zbir cifara) u globalnoj promenljivoj "suma".

Primer 3:

```
;
; Stampati zbirove cifara svih brojeva od 1 do nekog broja koji se unosi sa tastature.
;
#include <stdio.h>
;
int suma;
;
void suma_cifara(int broj)
{
    int cifra, kolicnik = broj;
    ;
    suma = 0;
    while (kolicnik>0)
    {
        cifra = kolicnik%10;
        suma += cifra;
        kolicnik /= 10;
    }
}
;
int main()
{
    int broj=1, granica;
    ;
    printf("Unesite granicu: ");
    scanf("%d", &granica);
    ;
    while (broj<=granica)
    {
        suma_cifara(broj);
        printf("Suma cifara broja %d je %d\n", broj, suma);
        broj++;
    }
    ;
    return 0;
}
;
extern printf, scanf
segment .data
    poruka1      db      "Unesite granicu: ", 0
    poruka2      db      "Zbir cifara broja %u je %u", 10, 0
    fmt          db      "%u",0
    broj         dd      1 ; pocetna vrednost je 1

segment .bss
    granica      resd    1      ; gornja granica

segment .text
    global  main
main:
    enter    0,0                ; rutina za inicijalizaciju

    ; Ovde pocinje koristan kod
    push    poruka1
    call    printf              ; printf("Unesite broj: ");
    add     esp, 4

    push    granica
    push    fmt
    call    scanf               ; scanf("%u", &granica );
    add     esp, 8

while_petlja:
    ; while ( broj <= granica )
    mov     eax, [broj]
    cmp     eax, [granica]      ; uporedi [broj] i [granica]
    jnb     end_while_petlja    ; izlazi iz petlje ako je !(broj <= granica)

    mov     eax, [broj]          ; stavi [broj] u eax da bi ga odatle preuzela rutina suma_cifara
    call    suma_cifara         ; pozovi subrutinu koja racuna sumu cifara broja na steku

    push    ecx                 ; u ecx se nalazi rezultat (zbir cifara broja)
    push    dword [broj]
    push    poruka2
    call    printf
```

```

add     esp, 12

inc     dword [broj]      ; uvecaj [broj] za 1
jmp     while_petlja      ; skok na pocetak while petlje

end_while_petlja:
mov     eax, 0             ; vrati se nazad u C
leave
ret

;
; Potprogram za racunanje zbira cifara nekog broja
; Kao argument uzima broj smesten u eax registar, a rezultat vraca u registru ecx
; Cuvamo na steku stanje svih registara OSIM ecx u kome zelimo da vratimo rezultat!
;

segment .text
suma_cifara:
push    eax
push    ebx
push    edx
push    esi
push    edi
mov     ecx, 0             ; u ecx je suma cifara
while_petlja_sub:         ; while ( broj > 0 )
cmp     eax, 0
jna     end_while_petlja_sub ; izlazi iz petlje ako je !(broj > 0)
; edx:eax / 10 = eax sa ostatkom u edx
mov     edx, 0
mov     ebx, 10            ; u ebx registar se unese delilac 10
div     ebx                ; deljenje edx:eax / ebx = eax sa ostatkom edx
add     ecx, edx           ; ostatak, tj. cifra u edx registru dodaje se postojećoj sumi u ecx
jmp     while_petlja_sub   ; skok na pocetak while petlje
end_while_petlja_sub:
pop     edi
pop     esi
pop     edx
pop     ebx
pop     eax
ret                        ; povratak u glavni program

```