Arhitektura računara 1

ASEMBLER - 4. termin

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);
;
;       i=1;
;       while (i<=broj)
;       {
;              faktorijel *= i;
;              i++;
;       }
;
;       printf("Faktorijel broja %d je %d.\n", broj, faktorijel);
;
;       return 0;
;}
;
%include "asm_io.inc"

segment .data
       poruka1        db      "Unesite broj: ", 0
       poruka2        db     "Faktorijel broja ", 0
       poruka3        db     " je ", 0
       faktorijel     dd     1


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


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

       ; Ovde pocinje koristan kod
       mov    eax,  poruka1
       call    print_string

       call    read_int             ; scanf("%u", &broj );
       mov     [broj], eax

       mov     dword [i], 1       ; broj = 1

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

       mov    eax, [faktorijel]         ; eax = faktorijel
       mul    dword [i]                 ; eax *= i
       mov    [faktorijel], eax         ; faktorijel = edx:eax (visih 32 bita koji se nalaze u edx
se zanemaruju)


       inc    dword [i]                 ; i++
```

```
        jmp     while_petlja       ; skok na pocetak while petlje
end_while_petlja:

        mov     eax, poruka2
        call    print_string

        mov     eax, [broj]
        call    print_int

        mov     eax, poruka3
        call    print_string

        mov     eax, [faktorijel]
        call    print_int
        call    print_nl

        popa
        mov     eax, 0              ; vrati se nazad u C
        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;
;}
;
%include "asm_io.inc"

segment .data
        poruka1        db     "Unesite broj: ", 0
        poruka2        db     "Zbir cifara broja ", 0
        poruka3        db     " je ", 0

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

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

        ; Ovde pocinje koristan kod
        mov     eax,  poruka1
        call    print_string

        call    read_int             ; ucitavanje, kao scanf("%u", &broj ); u C-u
        mov     [broj], eax
        mov     ecx, 0               ; u ecx ce se nalaziti suma cifara, inicijalizacija na nulu

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 postojecoj sumi u ecx
        jmp while_petlja             ; skok na pocetak while petlje
end_while_petlja:

        mov eax, poruka2
        call print_string

        mov eax, [broj]
        call print_int

        mov eax, poruka3
        call print_string

        mov eax, ecx
        call print_int
        call print_nl
```

```
popa
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 ma stek i skidaju sa njega.

Sledi primer čiji je zadatak da učita 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;
;}
;
%include "asm_io.inc"

segment .data
       poruka1        db      "Unesite granicu: ", 0
       poruka2        db      "Zbir cifara broja ", 0
       poruka3        db      " je ", 0
       broj           dd      1

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

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

       ; Ovde pocinje koristan kod

       mov     eax,  poruka1         ; stampaj poruku za unos granice
       call    print_string

       call    read_int              ; scanf("%u", &broj );
       mov     [granica], eax
while_petlja:                        ; while ( broj <= granica )
       mov     eax, [broj]
       cmp     eax, [granica]        ; uporedi [broj] i [granica]
       jnbe    end_while_petlja      ; izlazi iz petlje ako je !(broj <= granica)
       mov     eax, poruka2          ; stampaj poruke
       call    print_string
       mov     eax, [broj]           ; stampanje vrednosti [broj]
       call    print_int
       mov     eax, poruka3
       call    print_string
       mov     eax, [broj]           ; stavi [broj] u eax registar
       call    suma_cifara           ; pozovi subrutinu koja racuna sumu cifara broja u eax
       mov     eax, ecx              ; subrutina vraca vrednost sume cifara u registru ecx
```

```
       call    print_int             ; stampaj sumu cifara aktuelnog broja
       call    print_nl
       inc     dword [broj]          ; uvecaj [broj] za 1
       jmp     while_petlja          ; skok na pocetak while petlje
end_while_petlja:
       popa
       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 globalnoj promenljivoj "suma"
;
;

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                ; setuje se samo 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 dodaje se postojecoj 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
```