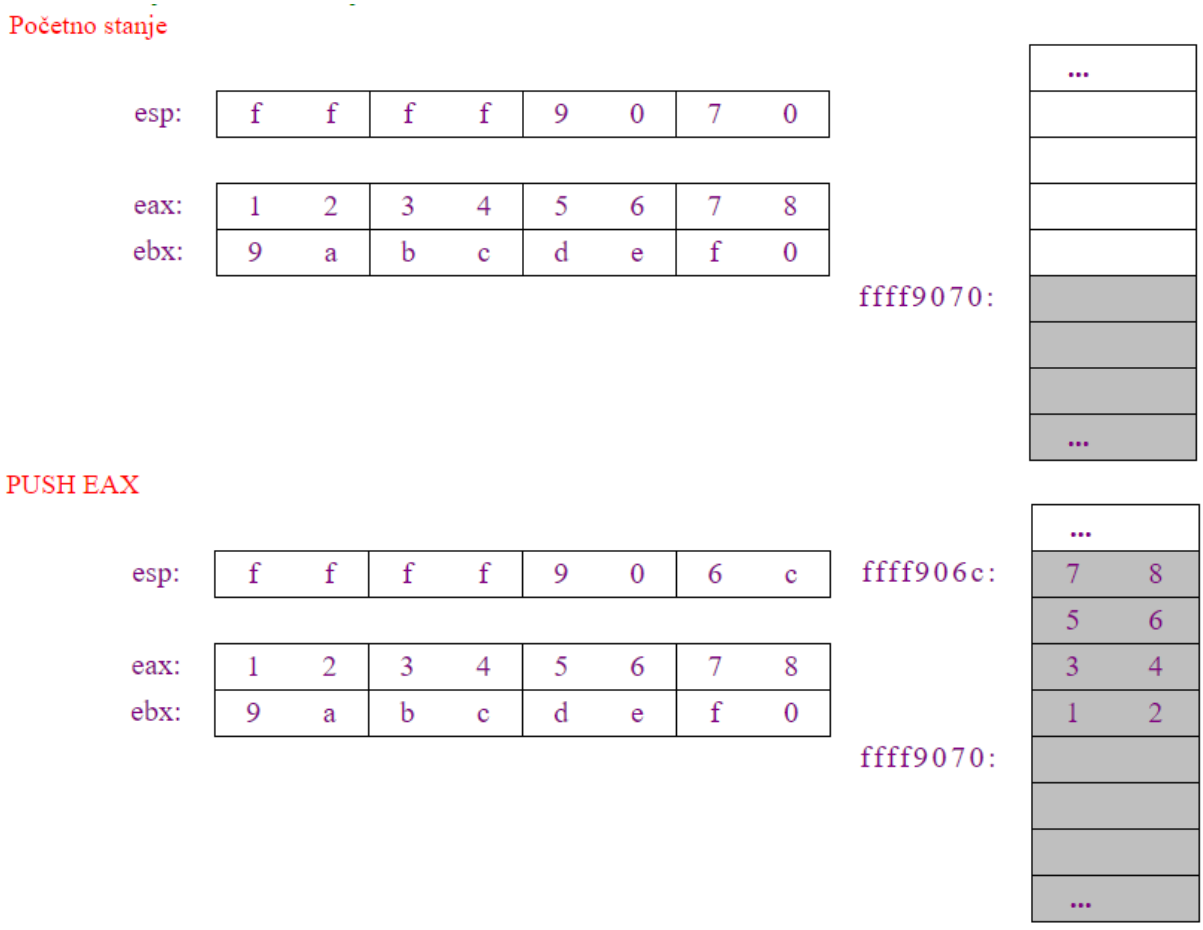
Stek memorija

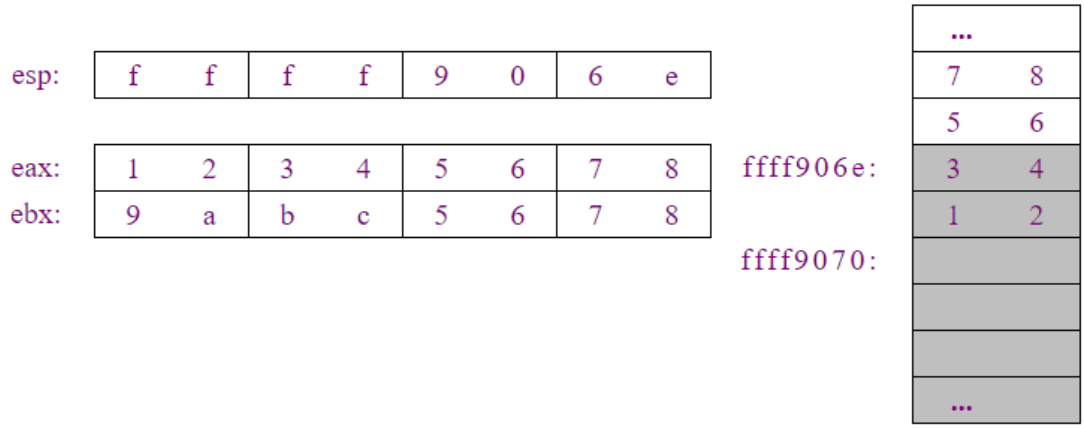
Stek memorija je bezadresna memorija sa sekvencijalnim pristupom. U svrhu realizacije steka, često se koristi operativna memorija. Baziran je na principu LIFO (Last In First Out), što znači da se sa steka prvo skida poslednji unet podatak. Prilikom pristupa stek memoriji, mi možemo izvršiti postavljanje ili preuzimanje podataka. Na samom početku je potrebno specificirati veličinu memorijskog prostora odvojenog za stek. Nadalje, potrebno je postaviti segmentni registar SS na segmentnu adresu početka steka, tj. registar SS treba da pokazuje na početak prostora u memoriji (segmenta) u kom se stek nalazi (SS je tada stek segment, odnosno selektor steka). Na kraju, treba postaviti registar ESP (odnosno SP kod 16-bitnih segmenata) tako da ukazuje na vrh steka. Taj registar neće sadržati apsolutnu adresu vrha steka, već vrednost offset-a u odnosu na početak memorijskog segmenta koji je odvojen za stek (tj. offset-a u odnosu na vrednost registra SS). Dakle, ESP će pokazivati na poslednji element (reč ili dvostruku reč) koja je uneta na stek.

Instrukcije za postavljanje podataka na stek i skidanje podataka sa steka su:
PUSH S - $S \rightarrow stek$. Izvorni operand se smešta na stek. Argumenat ove instrukcije je reč ili dvostruka reč, a počev od procesora 80386 kod ove instrukcije može figurisati i neposredno adresiranje.
POP D - $stek \rightarrow D$. Vrednost sa vrha steka se smešta u odredišni operand. Argumenat ove instrukcije je reč ili dvostruka reč.

Primer



POP BX



Primećuje se da se pri stavljanu podataka na stek, vrednost ESP smanjuje, a da se pri skidanju sa steka vrednost ESP povećava. Izvršenje instrukcije PUSH započinje sa umanjenjem sadržaja ESP za potrebnu veličinu, pa se tek potom vrši prenos u memoriju. Kod instrukcije POP se prvo vrši prenos podataka, a potom sa uvećava ESP.

PUSHAD (eng. push all doublewords - gurni sve dvostruke reči) - $opšti\ registar \rightarrow stek$. Sadržaj opštih registara se gura na stek. Ova instrukcija nema operanada. Ona postoji kod procesora 80386 i njegovih sledbenika (potomaka).

POPAD (eng. pop all doublewords - skini sve dvostruke reči) - $stek \rightarrow opšti\ registri$. Vrednosti sa steka se smeštaju u opšte registre. Ni ova instrukcija nema operande. Ta instrukcija nije postojala kod prethodnika procesora 80386.

Instrukcija PUSHAD će registre na stek gurnuti po sledećem redosledu: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI, pri čemu se na stek gura vrednost koju je ESP imao pre izvršenja instrukcije PUSHAD. Instrukcija POPAD će, naravno, vrednosti sa steka skidati u obrnutom redosledu.

PUSHA (eng. push all - gurni sve) - $16-bitni\ opšti\ registar \rightarrow stek$. Sadržaj opštih registara se gura na stek. Ova instrukcija nema operanada.

POPA (eng. pop all - skini sve) - $stek \rightarrow 16-bitni\ opšti\ registri$. Vrednosti sa steka se smeštaju u opšte registre. Ni ova instrukcija nema operande.

Instrukcije PUSHA i POPA izvršavaju ekvivalentne operacije kao PUSHAD i POPAD, samo sa 16-bitnim registrima, a osnovni razlog njihovog postojanja je kompatibilnost sa prethodnicima procesora 80386.

Zadatak 1.

Učitava se broj i ispisuje se zbir cifara broja. Zbir cifara broja se izračunava u posebnom potprogramu.

```
%include "asm_io.inc"
segment .data
;
; initialized data is put in the data segment here
;
msg1 db "Unesite broj: ",0
msg2 db "Zbir cifara broja ",0
msg3 db " je ",0

segment .bss
;
; uninitialized data is put in the bss segment
```

```
;
broj resd 1


segment .text
    global  _asm_main
_asm_main:
    enter   0,0               ; setup routine
    pusha

      mov eax, msg1
      call print_string
      call print_nl

      call read_int
      mov [broj], eax

      mov eax, msg2
      call print_string
      mov eax, [broj]
      call print_int
      mov eax, msg3
      call print_string
      mov eax, [broj]
      call suma_cifara
      mov eax, ecx
      call print_int
      call print_nl


    popa
    mov     eax, 0            ; return back to C
    leave
    ret


segment .text
suma_cifara:
      push eax
      push ebx
      push edx
      mov ecx, 0   ; u ecx je suma cifara
while_petlja_sub:
      cmp eax, 0
      jna end_while_petlja_sub
      mov edx, 0
      mov ebx, 10
      div ebx
      add ecx, edx
      jmp while_petlja_sub
end_while_petlja_sub:
      pop edx
      pop ebx
      pop eax
      ret   ;povratak u glavni program
```