8. Nastavni modeli x86 procesora - 1

Primer 1 - Jednostavan program bez I/Opodataka

Sledeći pro gram EX1.x86 izračunava vrednost izraza AX:=3*(a+b-1) gde su a i b simbolička imena za memorijske reči sa adresa 1000h i 1002h.

```
mov ax, [1000]      ; upisuje podatak sa memorijske lokacije 1000h u registar AX
mov bx, [1002]
add ax, bx          ; AX:=AX+BX
sub ax, 1           ; AX:=AX-1
mov bx, ax
add bx, ax
add ax, bx          ; AX:=3*AX
halt                ; zaustavlja izvrsavanje programa
```

1. Napisati program za SIMx86 koji izračunava vrednost izraza:

a) AX:= 2*(2a+3b-c) b) BX:=2a-b+3c

gde su a, b i c simbolička imena za memorijske reči koje se nalaze na adresama 1000h, 1002h i 1004h.

Primer 2 - Jednostavan program sa I/O instrukcijama get i put

Sledeći program EX2.x86 učitava niz celih brojeva koje upisuje se na uzastopne memorijske lokacije počev od adrese 1000h, a zatim izračunava i prikazuje njihov zbir. Unos vrednosti 0 znači da nema više ulaznih podataka.

```
        mov bx, 1000
a:      get
        mov [bx], ax
        add bx, 2
        cmp ax, 0
        jne  a
        mov cx, bx
        mov bx, 1000
        mov ax, 0
b:      add ax, [bx]
        add bx, 2
        cmp bx, cx
        jb  b
        put
        halt
```

2. Napisati program za SIMx86 koji za ulazne podatke x1, x2 i x3 izračunava i prikazuje vrednost izraza:

a) y = max(x1, x2, x3)

b) y = 1, ako su sva tri broja različita
   y = 2, ako su tačno dva među brojevima x1, x2, x3 međusobno jednaka
   y = 3, ako su sva tri broja međusobno jednaka.

Primer 3 - Program koji ilustruje memorijski mapiran I/O

Sledeći program EX3.x86 prihvata ulazne podatke - vrednosti dve logičke promenljive sa prekidača povezanih sa memorijskim lokacijama 0FFF0h i 0FFF2h i za njih izračunava vrednosti logičkih funkcija AND, OR, XOR i EQU (ekvivalencije) koje prikazuje preko LED dioda povezanih sa memorijskim lokacijama 0FFF8h, 0FFFAh, 0FFFCh i 0FFFEh. U trenutku kada se uključi treći prekidač, povezan sa lokacijom 0FFF4h, program prekida sa radom.

```
a:      mov ax, [fff0]
        mov bx, [fff2]
        mov cx, ax              ; Izracunavanje Sw0 AND Sw1
        and cx, bx
        mov [fff8], cx
        mov cx, ax              ; Izracunavanje Sw0 OR Sw1
        or cx, bx
        mov [fffa], cx
        mov cx, ax              ; Izracunavanje Sw0 XOR Sw1
        mov dx, bx              ; A XOR B = AB' + A'B
        not cx
        not bx
        and cx, bx
        and dx, ax
        or cx, dx
        mov [fffc], cx
        not cx                  ; Izracunavanje Sw0 EQU Sw1
        mov [fffe], cx          ; EQU = NOT XOR
        mov ax, [fff4]          ; Cita stanje treceg prekidaca
        cmp ax, 0               ; Proverava da li je ukljucen
        je a                    ; Ponavlja izracunavanja ako je iskljucen
        halt
```

3. Napisati program za SIMx86 koji prihvata vrednosti dve logičke promenljive sa prekidača povezanih sa memorijskim lokacijama 0FFF0h i 0FFF2h i za njih izračunava vrednosti logičkih funkcija ⇨ (implikacija), ⇦, NAND i NOR koje prikazuje preko LED dioda povezanih sa memorijskim lokacijama 0FFF8h, 0FFFAh, 0FFFCh i 0FFFEh. U trenutku kada se uključi treći prekidač, povezan sa lokacijom 0FFF4h pro gram prekida sa radom.

4. Napisati program za SIMx86 koji prihvata vrednosti tri logičke promenljive - prenosa iz prethodnog razreda (p_{i-1}) i dveju binarnih cifara (x_i, y_i) sa prekidača povezanih sa memorijskim lokacijama 0FFF0h, 0FFF2h i 0FFF4h i za njih izračunava vrednosti logičkih funkcija punog sabirača: vrednost zbira na i-toj poziciji ($z_i=x_i + y_i + p_{i-1}$) i vrednost prenosa u sledeći razred (p_i) koje prikazuje preko LED dioda povezanih sa memorijskim lokacijama 0FFF8h i 0FFFAh. Uključivanje četvrtog prekidača 0FFF6h prekida rad programa.

5. Napisati program za SIMx86 koji prihvata vrednosti dva dvocifrena binarna broja sa prekidača povezanih sa memorijskim lokacijama (FFF0/FFF2) i (FFF4/FFF6) i prikazuje njihov trobitni zbir (dvobitni rezultat plus prenos) na LED diodama FFF8, FFFA, FFFC. Koristite logičke funkcije za pun sabirač iz prethodnog zadatka. Nemojte koristiti ADD instrukciju, već prob lem rešite korišćenjem logičkih instrukcija.

9. Nastavni modeli x86 procesora - 2

1. Napisati pro gram za SIMx86 koji prihvata n - broj članova niza i upisuje ga na adresu 1000, a zatim i n članova niza koje upisuje na adrese počev od 1002 (n <= 20) i na kraju

   a) prebrojava koliko je među njima manjih od 10 i prikazuje taj rezultat;

   b) prebrojava koliko je među njima parnih i prikazuje taj rezultat

   Uputstvo: AND AX,1 daće kao rezultat 0 ako je broj paran, jer se završava sa 0b;

   c) izračunava njihov zbir i prikazuje ga,

   d) utvrđuje koji je među njima najmanji i prikazuje ga;

   e) utvrđuje koji među njima ima najmanju apsolutnu vrednost i prikazuje ga (broj, a ne njegovu apsolutnu vrednost).

2. Skup instrukcija x86 ne sadrži naredbu za množenje. Napišite program MUL.x86 koji učitava dve vrednosti i izračunava i prikazuje njihov proizvod.

Uputstvo - množenje se može predstaviti kao ponovljeno sabiranje.

3. Skup instrukcija x86 ne sadrži naredbu za celobrojno deljenje. Napišite program DIV.x86 koji učitava dve vrednosti i izračunava i prikazuje njihov količnik.

Uputstvo - celobrojno deljenje se može predstaviti kao ponovljeno oduzimanje.

4. Skup instrukcija x86 ne sadrži naredbu za izračunavanje ostatka pri celobrojnoj deobi. Napišite pro gram MOD.x86 koji učitava dve vrednosti x i y i izračunava i prikazuje x mod y.

5. Skup instrukcija x86 ne sadrži naredbe za linijska pomeranja. Napišite programe za izvršavanje sledećih operacija:

   a) SHL AX,1 b) SHR AX,1

Uputstvo: add ax,ax vrši SHL ax,1. Celobrojno deljenje sa 2 daće rezultat SHR.

6. Skup instrukcija x86 ne sadrži naredbe za ciklička pomeranja. Napišite programe za izvršavanje sledećih operacija:

   a) ROL AX,1 b) ROR AX,1

Uputstvo: Za realizaciju ROL možete testirati najstariji bit proverom da li je vrednost u ax veća od 8000h i dodavati ga na vrednost dobijenu posle SHL.

Primer 4 - Program koji ilustruje DMA

Sledeći program EX4.x86 startuje od labele d upisivanjem 0 na lokaciju 1000. Zatim ulazi u petlju u kojoj proverava dva uslova - da li je uključen prekidač FFF0 i da li je korisnik promenio vrednost na lokaciji 1000. Uključivanje prekidača prekida izvršavanje programa, a promena vrednosti na lokaciji 1000 predaje upravljanje na na sekciju sa labelom c u kojoj se sabira vrednost n reči gde je n nova vrednost sa lokacije 1000. Program sabira vrednosti sa uzastopnih lokacija počev od adrese 1002, prikazuje zbir i vraća upravljanje na labelu d.

```
d:      mov cx, 0               ;Brise lokaciju 1000h pre no sto je testiramo
        mov [1000], cx
                                ;U sledecoj petlji proveravamo da li je promenjen sadrzaj
                                ; sa lokacije 1000h  i da li je ukljucen prekidac FFF0
a:      mov cx, [1000]
        cmp cx, 0
        jne  c                  ; Skok ako je promenjena vrednost na lokaciji 1000h
        mov ax, [fff0]          ; Ako nije citamo stanje prekidaca FFF0
        cmp ax, 0
        je  a                   ; Skok ako je iskljucen
        halt                    ; Ako je ukljucen, prekida se izvrsavanje programa
                                ; Sledeci fragmaent sabira “cx” uzastopnih reci od kojih je prva
                                ; na adresi 1002h, a po zavrsenom sabiranju prikazuje zbir.
c:       mov bx, 1002           ;Inicijalizuje BX da pokazuje na pocetak niza
        mov ax, 0               ;Inicijalizuje zbir
b:      add ax, [bx]            ;Dodaje sledeci podatak.
        add bx, 2               ;pomera pokazivac BX na sledeci elemnt niza
        sub cx, 1               ;Smanjuje brojac
        cmp cx, 0
        jne  b
        put                     ;Prikazuje zbir i vraca se na pocetak
        jmp  d
```

Primer 5 - Program koji ilustruje kori{}enje sistema prekida

Sledeći pro gram sastoji se iz dva mod ula - EX5a.x86 i EX5b.x86. Glavni pro gram EX5a.x86 poredi memorijske lokacije 1000h i 1002h. Ako njihovi sadržaji nisu jednaki glavni program prikazuje vrednost sa adrese 1000h, kopira tu vrednost na lokaciju 1002h i ponavlja ceo proces sve dok se ne uključi prekidač FFF0.

```
a:      mov ax, [1000]          ;Uzima podatak sa lokacije1000h
        cmp ax, [1002]          ;Poredi fga sa podatkom na lokaciji 1002h
        je  b                   ;Ako su jednaki proverava stanje prekidaca FFF0h.
        put                     ;Ako su razliciti prikazuje vrednost sa lokacije 1000h (AX)
        mov [1002], ax          ;upisuje prikazanu vrednost na 1002h
b:      mov ax, [fff0]          ;Testira prekidac FFF0
        cmp ax, 0
        je  a
        halt
```

Rutina za opsluživanje prekida (in ter rupt ser vice rou tine - ISR EX5b.x86) nalazi se na lokaciji 100h, za razliku od glavnog programa koji se nalazi na adresi 0. Kada se primi sig nal za prekid, odgovarajuća ISR uvećava vrednost na adresi 1000h i vraća upravljanje glavnom programu.

```
        mov ax, [1000]          ;Uvecava sadrzaj sa adrese  lo ca tion 1000h za 1
        add ax, 1
        mov [1000], ax
        iret                    ;vraca upravljanje prekinutom programu
```

10. Nastavni modeli x86 procesora - 3

Primer 6 - Program koji ilustruje samomodifikaciju

Sledeći pro gram EX6.x86

```
        sub ax, ax
        mov [100], ax       ;Postavlja 0 na lokaciju 100h. Promena ove vrednosti
a:      mov ax, [100]       ;bice indikator da je izvrsen novogenerisan kod
        cmp ax, 0
        je  b               ;Ako je [100h]=0, prelazi se na generisanje koda
        halt                ;U suprotnom se zavrsava sa radom programa
b:      mov ax, 00c6        ;Pocev od ove naredbe, upisuju se kodovi masinskih
        mov [100], ax        instrukcija u memoriju pocev od lokacije 100h
        mov ax, 0710
        mov [102], ax
        mov ax, a6a0
        mov [104], ax
        mov ax, 1000
        mov [106], ax
        mov ax, 8007
        mov [108], ax
        mov ax, 00e6
        mov [10a], ax
        mov ax, 0e10
        mov [10c], ax
        mov ax, 4
        mov [10e], ax
        jmp 100             ;predaje upravljanje generisanom kodu
```

Upisuje sledeći kod na lokaciju 100 i izvršava ga.

```
        mov ax, [1000]      ;Generisan kod prikazuje broj sa lokacije 1000h
        put
        add ax, ax
        add ax, [1000]
        put                 ; i njegovu trostruku vrednost
        sub ax, ax
        mov [1000], ax      ;a zatim upisuje 0 na lokaciju 1000h
        jmp 0004            ;0004 je adresa labele a na koju se vraca upravljanje
```

Primer 7 - Samomodifikacija koja realizuje poziv procedure

Skup instrukcija hipotetičkih procesora ne sadrži naredbu za poziv procedure. Međutim programi EX7a.x86 i EX7b.x86 simuliraju poziv potprograma i povratak upravljanja glavnom programu korišćenjem samomodifikacije. Program prevodi neoznačen ceo dekadni broj u binarni brojni sistem. Očekuje broj u AX, prevodi ga u niz nula i jedinica koje upisuje na uzastopne memorijske lokacije počev od adrese 1000h.

```
                ; Potprogram koji treba upisati u memoriju pocev od adrese 100h
        mov bx, 1000        ;pocetna adresa stringa
        mov cx, 10          ;brojac=16 (10h), tj. broj bitova u reci
a:      mov dx, 0           ;pretpostavljamo da je tekuci bit 0
        cmp ax, 8000        ;proveravamo da li je H.O. bit 0 ili 1
        jb b
        mov dx, 1           ;ako je  H.O. bit u AX jednak 1
b:      mov [bx], dx        ;upisujemo 0 ili 1 na tekucu memorijsku lokaciju.
        add bx, 1           ;pomeramo pointer BX na sledecu lokaciju
        add ax, ax          ;AX = AX *2 (shift left)
        sub cx, 1           ;
        cmp cx, 0           ;ponavljamo 16 puta
        ja a
        jmp 0       ;povratak upravljanja glavnom programu koji pre poziva  potprograma
                    ; na adresni deo ove instrukcije upisuje adresu povratka.
```

Jedina instrukcija koju program modifikuje u potprogramu je poslednja naredba jmp. Ovaj skok treba da vrati kontrolu na instrukciju neposredno iza JMP kojim je predato upravljanje potprogramu. Da bi se to omogućilo glavni program treba da upamti adresu povratka na mestu argumenta u poslednjoj instrukciji sledećeg koda. Ako kod počinje na lokaciji 100h odgovarajuća JMP instrukcija biće na adresi 120h. Dakle, glavni program adresu povratka mora da upiše na adresu 121h.

```
                ;Glavni program koji treba upisati pocev od adrese 0
        mov ax, 000c        ;Adresa prve instrukcije BRK
        mov [121], ax       ;upis adrese povratka (prve instrukcije BRK) adr. deo JMP
        mov ax, 1234        ;Predaja broja 1234h koji ce biti preveden u binarni
        jmp 100             ;"Call" potprograma
        brk                 ;Pauza da korisnik proveri sadrzaj niza iz memorije
        mov ax, 0019        ;Adresa sledece BRK instrukcije
        mov [121], ax
        mov ax, fdeb        ;Prevodjenje broja 0FDEBh u binarni zapis.
        jmp 100
        brk
        mov ax, 26
        mov [121], ax
        mov ax, 2345        ;Prevodjenje broja  2345h u binarni zapis.
        jmp 100
        halt
```

U slede}im zadacima koristiti programsku samomodifikaciju za realizaciju poziva procedure i povratka u glavni program.

1. Napisati proceduru za SIMx86 koja nalazi i štampa maksimum niza reči koje su upisane u memoriju počev od lokacije 1000h. Broj elemenata niza nalazi se u registru CX. Glavni program treba da poziva ovu proceduru više puta sa različitim vrednostima CX.

2. Napisati proceduru za SIMx86 koja očekuje odresu u BX, broj u CX i vrednost u AX. Procedura treba da napravi CX kopija broja AX počev od adrese BX. Glavni program ovu proceduru treba da poziva tri puta sa različitim vrednostima u BX, CX i AX koje treba da uzima sa ulaza.

11. Nastavni modeli x86 procesora - 4

1. Napisati potprograme za MUL, DIV i MOD koji bi se iz glavnog programa pozivali korišćenjem samomodifikacije. Prenos ulaznih parametara ostvariti kroz registre AX i BX, a rezultat ostaviti u CX registru.

2. Napisati pro gram za izračunavanje i izdavanje vrednosti izraza (-(a mod b)+(a div b)*c) koji za računanje vrednosti proizvoda, količnika i ostatatka poziva potprograme iz prethodnog zadatka.

3. Napisati potprograme za SHL, SHR, ROL i ROR vrednosti bilo kojeg procesorskog registra za jednu poziciju koji bi se iz glavnog programa pozivali korišćenjem samomodifikacije.

4. Napisati pro gram DA TUM.x86 koji sa ulaza prihvata tri podatka: dan, mesec, godina i na osnovu njih formira i prikazuje pakovani datum dddddmmmmgggggggg gggg u binarnom zapisu (Vidi zadatak 15. u 8. vežbanju).

5. Napisati program LIGHTSHOW.x86 program koji prikazuje "lajt šou" na SIMx86 LED diodama. To se može ostvariti upisivanjem niza vrednosti u odgovarajuće memorijske lokacije sa malom zadrškom koja se ostvaruje "praznim" petljama. Vrednosti koje treba prikazivati na LED diodama smestiti u memoriju i odatle ih uzimati u ciklusu koji treba ponavljati sve dok ne bude uključen prekidač FFF0.

6. Napisati program BCD.x86 koji za unet prirodan "8421" kod dekadne cifre 0..9 preko prekidača FFF0..FFF6 prikazuje na LED diodama FFF8..FFFE njen:

a) Stibitzov "višak 3" kod;

b) Aikenov "2421" kod;

c) Grejov ciklični kod;

Uputstvo: Tabele odgovarajućih kodova date su vežbi 3.

7. Napisati program koji uređuje reči sa memorijskih lokacija 1000h..1020h u neopadajući redosled korišćenjem sledećih algoritama sortiranja:

a) *Metoda razmene* koja se obavlja razmenjivanjem a[1] sa a[j] (j=2,3,...,n) koje je manje od a[1]; razmenjivanjem a[2] sa a[j] (j=3,4,...,n) koje je manje od a[2]. Isti postupak primenjuje se na preostale elemente osim poslednjeg koji mora da se nalazi na odgovarajućem mestu. Odgovarajući kod u Pascalu je:

```
procedure Sort1(n:integer;var a:niz);
        var i,j:integer;
        begin
                for i:=1 to n-1 do
                        for j:=i+1 to n do if a[i]>a[j] then razmeni(a[i],a[j])
        end;
```

b) *Selection sort* koji podrazumeva da treba minimalni element niza razmeniti sa a[1], minimalni element odsečka a[2], a[3],...,a[n] razmeniti sa a[2], minimalni element odsečka a[3], a[4],...,a[n] razmeniti sa a[3]; isti postupak primeniti na preostale elemente osim

poslednjeg koji se nalazi na svom mestu. Odgovarajući kod u Pascalu je:

```
procedureSort2(n:integer;a:niz);
        vari,j,MinInd:integer;
        begin
          for i:=1 to n-1 do
            be gin
                    MinInd:=i;{*odredjivanje indeksa minimalnog elementa*}
                    for j:=i+1 to n do{*medju elementima: a[i],...,a[n]*}
                    if a[i]<a[MinInd] then MinInd:=i;
                    if i<>MinInd then razmeni(a[i],a[MinInd])
             end
        end;
```

c) *Bubble sort* koji se sastoji u tome da susedni elementi niza razmenjuju vrednosti ukoliko nisu u traženom poretku, time se najveći el e ment prebaci na svoje mesto (kraj niza); zatim se isti postupak primeni na svim elementima niza osim poslednjeg, itd. Odgovarajući kod u Pascalu je:

```
procedureSort3(n:integer;vara:niz);
        vari,granica:integer;
        begin
                    for granica:=n-1 downto 1 do
                      for i:=1 to granica do
                                if a[i]>a[i+1] then razmeni(a[i],a[i+1])
        end;
```