Skupovi instrukcija za Intel

Vladimir Filipović
vladaf@matf.bg.ac.yu

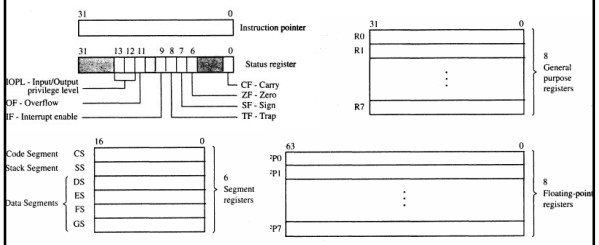
Registri, memorija i prenos podataka kod IA-32 procesora

Korporacija Intel koristi generičko ime Intel Arhitektura (IA) za procesore koje proizvodi. Ovde će biti opisani IA procesori koji operišu sa 32-bitnim memorijskim operandima i 32-bitnim operandima-podacima. Oni se označavaju kao IA-32 procesori.
Prvi IA-32 procesor je 80386, koji se pojavio 1985. Od tada su nastali 80486 (1989), Pentium (1993), Pentium Pro (1995), Pentium II (1997), Pentium III (1999) i Pentium IV (2000). Ovi procesori se karakterišu sve boljim performansama, koje su postignute pomoću velikog broja arhitektonskih poboljšanja i poboljšanja u mikroelektronskoj tehnologiji. Najnoviji članovi ove familije imaju specijalizovane instrukcije za rukovanje multimedijalnim i grafičkim podacima i za vektorsko procesiranje podataka.
Skup instrukcija za IA-32 je veoma velik. Ovde ćemo se ograničiti na osnovne instrukcije i osnovne adresne modove.

Kod IA-32 arhitekture, memorija je bajt-adresibilna i instrukcije operišu nad podacima dužine 8 ili 32. U Intel-ovoj terminologiji, ovi operandi se zovu bajt i dvostruka reč. Operand dužine 16 se naziva reč, jer je to bilo uobičajeno kod prethodnih 16-bitnih Intel procesora.

Registri, memorija i prenos podataka kod IA-32 procesora

Kod IA-32 procesora se koristi little endian adresiranje memorije.
Memorijski operandi koji sadrže više bajtova mogu da počnu od ma koje adrese u memoriji, tj. memorijske adrese ne moraju biti reč-poravnate.
Sledeća slika opisuje registre IA-32:



Registri, memorija i prenos podataka kod IA-32 procesora

Registri označeni sa R0 do R7 su **opšti registri**, koji (uz neke izuzetke) čuvaju bilo podatke, bilo adrese.
Postoji i osam **registara za rad sa brojevima u pokretnom zarezu**, koji sadrže bilo dvostruke, bilo četvorostruke reči. Ovi registri sadrže i proširenje, koje omogućuje da se tu smeste reči dužine 80 bitova.
Arhitektura IA-32 je zasnovana na memorijskom modelu koji različitim oblastima memorije (nazvanim segmenti) pridružuje različite svrhe korišćenja. Segment koda sadrži instrukcije programa. Segment steka sadrži stek procesora, a četiri segmenta podataka su obezbeđena radi čuvanja podataka programa. **Segmentni registri** sa prethodne slike sadrže selektorske vrednosti, koje mogu biti korišćene u lociranju ovih segmenata u memorijskom adresnom prostoru. Treba istaći da 32-bitna adresa kod IA-32 procesora obezbeđuje pristup ma kojoj memorijskoj lokaciji, bez obzira da li se radi o programu, procesorskom steku ili podacima programa.
Specijalizovani registri procesora IA-32 su **brojač naredbi** i **statusni registar**.

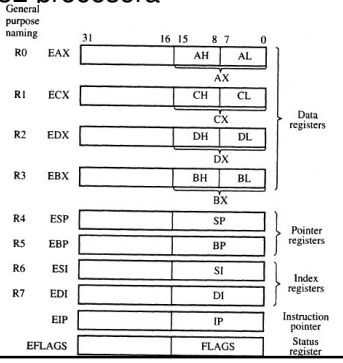
Registri, memorija i prenos podataka kod IA-32 procesora

IA-32 procesori su dizajnirani sa namerom obezbeđenja kompatibilnosti sa ranijim Intel procesorima (8086 i 80286), pa i njihovi opšti registri dopuštaju način rada koji karakteriše njihove prethodnike, kod kojih se moglo raditi samo sa reči ili sa bajtom i kod kojih su bila postavljena veća ograničenja na način korišćenja pojedinih registara.

Kod Intel-ovih 8-bitnih procesora, registri su se zvali A,B, C, D.
Kod 16-bitnih procesora, registri su bili AX,BX,CX, DX, pri čemu su viši i niži bajtovi tih registara bili identifikovani sa sufiksima H i L respektivno (umesto X). Tako je AH označavalo gornji bajt registra AX, a BL donji bajt registra BX.
Kod IA-32 nazivima registara se dodaje prefiks E, koji označava proširivanje, pa se stoga u Intel-ovoj tehničkoj dokumentaciji ovi registri nazivaju EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI.

Dizajn IA-32 procesora omogućuje da program napisan na mašinskom jeziku nekog starijeg 16-bitnog procesora korektno radi na IA-32 procesoru, ako se stanje procesora postavi u taj režim rada. Procesor IA-32 može dinamički da se prebacuje između 16-bitnih i 32-bitnih operacija na osnovu bajtova koji predstavljaju prefiks instrukcije.

Registri, memorija i prenos podataka kod IA-32 procesora



Adresni modovi kod IA-32 procesora

IA-32 procesori sadrže veliki i fleksibilan skup adresnih modova, dizajniranih tako da omoguće pristup bilo pojedinim članovima, bilo elementima nizova i listi.

Name	Assembler syntax	Addressing function
Immediate	Value	Operand = Value
Direct	Location	EA = Location
Register	Reg	EA = Reg that is, Operand = [Reg]
Register indirect	[Reg]	EA = [Reg]
Base with displacement	[Reg + Disp]	EA = [Reg] + Disp
Index with displacement	[Reg * S + Disp]	EA = [Reg] × S + Disp
Base with index	[Reg1 + Reg2 * S]	EA = [Reg1] + [Reg2] × S
Base with index and displacement	[Reg1 + Reg2 * S + Disp]	EA = [Reg1] + [Reg2] × S + Disp

Value = an 8- or 32-bit signed number
Location = a 32-bit address
Reg, Reg1, Reg2 = one of the general purpose registers EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, with the exception that ESP cannot be used as an index register
Disp = an 8- or 32-bit signed number, except that in the Index with displacement mode it can only be 32 bits.
S = a scale factor of 1, 2, 4, or 8

Adresni modovi kod IA-32 procesora

Instrukcije IA-32 procesora sa dva operanda imaju opšti oblik

OPkod dst, src

što predstavlja isto uređenje izvora i odredišta kao kod ARM procesora.

Pogledajmo efekat različitih adresnih modova kod instrukcije za prenos podataka MOV:

neposredno

```
MOV EAX, 25
MOV EAX, 3FA00H
MOV EAX, 1010101B
```

apsolutno (tj. direktno kod Intel-a)

```
MOV EAX, LOKACIJA
```

neposredno

```
NUMBER EQU 25
MOV EAX, NUMBER
```

direktno

```
MOV EAX, [LOKACIJA]
```

neposredno

```
MOV EBX, OFFSET LOKACIJA
```

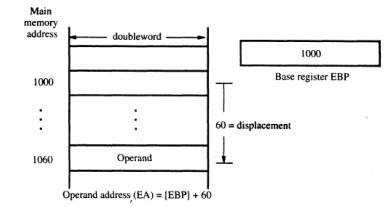

Adresni modovi kod IA-32 procesora

baza

```
MOV EAX, [EBX]
```

baza+pomeraj

```
MOV EAX,[EBP+60]
```

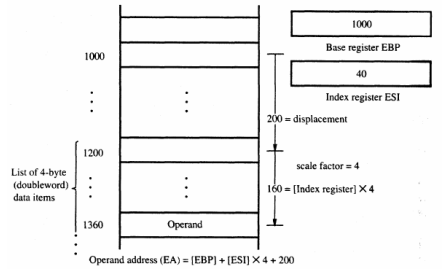



```
MOV AL, [EBP+10]
```


Adresni modovi kod IA-32 procesora

baza+(skalirani)indeks+pomeraj

```
MOV EAX,[EBP+ESI*4+200]
```



Iako se baza+pomeraj može predstaviti preko indeks+pomeraj, ona nije višak.

Instrukcije kod IA-32 procesora

Ovde će biti ukratko opisano samo nekoliko osnovnih instrukcija, koje će nam trebati za realizaciju programa primera:

Instrukcije sabiranja, prenos i oduzimanja imaju sledeći oblik:

ADD dst, src dst←[dst]+[src]

MOV dst, src dst←[src]

SUB dst, src dst←[dst]-[src]

Primer. Sledeći kod realizuje sabiranje sadržaja dvaju memorijskih lokacija A i B i smeštaj zbira u memorijsku lokaciju C:

```
MOV EAX, A
ADD EAX, B
MOV C, EAX
```


Instrukcije kod IA-32 procesora

Primer. Sledeća instrukcija uslovnog skoka :

```
JG LOOPSTART
```

Dovodi do uslovnog skoka na memorijsku lokaciju LOOPSTART ako je rezultat poslednje aritmetičke operacije bio veći od 0.

Sve instrukcije uslovnog prelaska počinju sa slovom J iza kog sledi oznaka uslova.

Sledeća tabela opisuje koji su uslovi indicirani kojom kombinacijom flegova:

		CF	ZF	SF	OF
Oznaćeni operandi	jednako	0	1	0	0
	manji	–	0	1	0
	manji	–	0	0	1
	veći	–	0	0	0
	veći	–	0	1	1
Neoznaćeni operandi	jednako	0	1	0	0
	ispod	1	0	–	–
	iznad	0	0	–	–

Instrukcije kod IA-32 procesora

Da bi se registar opšte namene iskoristio za registarski indirektno adresiranje, potrebno je da se prvo adresa memorijske lokacije kojoj se pristupa učita u registar. IA-32 instrukcije obezbeđuju dva načina za postizanje ovog cilja.

Ako je adresa eksplicitno poznata, npr. lokacija LOCATION, tada se ona može učitati korišćenjem instrukcije premeštanja i neposrednog adresnog moda:

MOV EBX, OFFSET LOCATION

Alternativno, može se koristiti i instrukcija za učitavanje efektivne adrese. U tom slučaju, instrukcija

LEA EBX, LOCATION ;dst←EA(src)

će imati isti efekat kao i prethodna instrukcija.

Instrukcija LEA može da se koristi za izračunavanje ma koje adrese koja se dinamički izračunava tokom izvršena programa

Instrukcije kod IA-32 procesora

INC dst

DEC dst

CMP dst, src

JMP dst

Program za sabiranje brojeva kod IA-32

```
          LEA     EBX,NUM1      Initialize base (EBX) and
          MOV     ECX,N           counter (ECX) registers.
          MOV     EAX,0         Clear accumulator (EAX)
          MOV     EDI,0           and index (EDI) registers.
STARTADD: ADD     EAX,[EBX + EDI *4]  Add next number into EAX.
          INC     EDI           Increment index register.
          DEC     ECX           Decrement counter register.
          JG      STARTADD      Branch back if [ECX] > 0.
          MOV     SUM,EAX       Store sum in memory.
```

(a) Straightforward approach

```
          LEA     EBX,NUM1      Load base register EBX and
          SUB     EBX,4           adjust to hold NUM1−4.
          MOV     ECX,N         Initialize counter/index (ECX).
          MOV     EAX,0         Clear the accumulator (EAX).
STARTADD: ADD     EAX,[EBX + ECX * 4]  Add next number into EAX.
          LOOP    STARTADD      Decrement ECX and branch
                                  back if [ECX] > 0.
          MOV     SUM,EAX       Store sum in memory.
```


Format mašinske instrukcije kod IA-32

OP code	Addressing mode	Displacement	Immediate
1 or 2 bytes	1 or 2 bytes	1 or 4 bytes	1 or 4 bytes

Asemblerski jezik kod IA-32

```
Assembler directives
          .data
          NUM1    DD    17, 3, -51, 242, -113
          N       DD    5
          SUM     DD    0

          .code

Statements that generate machine instructions
          MAIN :  LEA   EBX, NUM1
                  SUB   EBX, 4
                  MOV   ECX, N
                  MOV   EAX, 0
          STARTADD : ADD  EAX, [EBX+ECX * 4]
                  LOOP  STARTADD
                  MOV   SUM, EAX

Assembler directive
                  END   MAIN
```


Logičke instrukcije i pomeranje kod IA-32

Logičke instrukcije su AND, OR, XOR, NOT.
Instrukcija AND realizuje bitovnu konjukciju. Na primer, neka registar EAX sadrži heksadecimalni obrazac 0000FFFF, a EBX 02FA62CA. Izvršenjem instrukcije

AND EBX, EAX

će se u registar EBX postaviti heksadecimalni obrazac 000062CA.

Instrukcije pomeranja su SHL, SHR, SAL i SAR.

```
LEA     EBP,LOC       EBP points to first byte.
MOV     AL,[EBP]      Load first byte into AL.
SHL     AL,4          Shift left by 4 bit positions.
MOV     BL,[EBP+1]    Load second byte into BL.
AND     BL,0FH        Clear high-order 4 bits to zero.
OR      AL,BL         Concatenate the BCD digits.
MOV     PACKED,AL     Store the result.
```

Ulaz i izlaz kod IA-32

Izolovani ulaz i izlaz se tako označava da bi se razlikovao od memorijski mapiranog ulaza i izlaza, kod koga su adresibilne lokacije I/O uređaja u istom adresnom prostoru kao memorijske lokacije.
Kod IA-32 procesora, iste adresne linije i linije podataka se koriste za oba adresna prostora. Za indikaciju na koji adresni prostor referencira instrukcija koristi se izlazna kontrolna linija.

U bajt-adresibilnom I/O adresnom prostoru se koriste 16-bitne adrese.

IN reg, devadr

OUT devadr, reg ;(reg je AL ili EAX, a devadr je 8-bitna adresa uređaja)

IN reg, DX

OUT DX, reg ;(reg je AL ili EAX)

Ulaz i izlaz kod IA-32

```
        LEA     EBP,LOC         EBP points to memory area.
READ:   BT      INSTATUS,3      Wait for character to be
        JNC     READ              entered into DATAIN.
        MOV     AL,DATAIN       Transfer character into AL.
        MOV     [EBP],AL        Store the character in memory
        INC     EBP               and increment pointer.
ECHO:   BT      OUTSTATUS,3     Wait for display to
        JNC     ECHO              be ready.
        MOV     DATAOUT,AL      Send character to display.
        CMP     AL,CR           If not carriage return,
        JNE     READ              read more characters.
```

Izolovani ulaz i izlaz:

Ulaz i izlaz kod IA-32

Blokovski prenos se realizuje instrukcijama REPINS i REPOUTS.

Prefiks REP ukazuje na ponavljanje, a ponavlja se složenija forma instrukcije izolovanog prenosa. Sufiks S ukazuje da se radi o tzv. "string" operacijama.

Izvršavanje prethodnih instrukcija zavisi od sadržaja registara ECX i EDI.

Sufiks B ili D na op-kodu ovih mnemonika ukazuje da li se blok-instrukcija izvršava nad bajtovima ili nad dvostrukim rečima

Blokovski prenos:

```
LEA     EDI,MEMBLOCK
MOV     DX,DISKDATA
MOV     ECX,128
REPINSD
```


Potprogrami

PUSH src ; ESP ← ESP-4, [ESP] ←[src]

POP dst ; dst ← [ESP], ESP ← ESP+4

PUSHAD

POPAD

CALL LISTAADD

Potprogrami

Sledeći primer predstavlja ilustraciju realizovanja potprograma, gde su parametri preneseni preko registara.

```
Calling program
        :
        LEA     EBX,NUM1        Load parameters
        MOV     ECX,N             into EBX,ECX.
        CALL    LISTADD         Branch to subroutine.
        MOV     SUM,EAX         Store sum into memory.
        :
Subroutine
LISTADD:  PUSH  EDI             Save EDI.
          MOV   EDI,0           Use EDI as index register.
          MOV   EAX,0           Use EAX as accumulator register.
STARTADD: ADD   EAX, [EBX + EDI * 4]  Add next number.
          INC   EDI             Increment index.
          DEC   ECX             Decrement counter.
          JG    STARTADD        Branch back if [ECX] > 0.
          POP   EDI             Restore EDI.
          RET                   Branch back to Calling program.
```

ESP →

[EDI]
Return address
Old TOS

Sadržaj steka posle smeštanja EDI

Potprogrami

Sledeći primer predstavlja ilustraciju realizovanja potprograma, gde su parametri preneseni preko steka (glavni program).

```
(Assume top of stack is at level 1 below.)
Calling program
        PUSH    OFFSET NUM1     Push parameters onto the stack.
        PUSH    N
        CALL    LISTADD         Branch to the subroutine.
        ADD     ESP,4           Remove n from the stack.
        POP     SUM             Pop the sum into SUM.
        :
        :
```

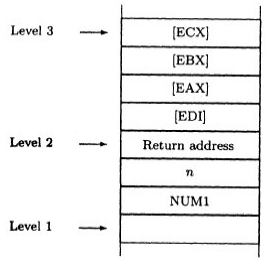
Potprogrami

Sledeći primer predstavlja ilustraciju realizovanja potprograma, gde su parametri preneseni preko steka (potprogram).

```
LISTADD:  PUSH   EDI              Save EDI and use
          MOV    EDI,0             as index register.
          PUSH   EAX              Save EAX and use as
          MOV    EAX,0             as accumulator register.
          PUSH   EBX              Save EBX and load
          MOV    EBX,[ESP+20]      address NUM1.
          PUSH   ECX              Save ECX and
          MOV    ECX,[ESP+20]      load count n.
STARTADD: ADD    EAX,[EBX+EDI*4]   Add next number.
          INC    EDI               Increment index.
          DEC    ECX               Decrement counter.
          JG     STARTADD          Branch back if not done.
          MOV    [ESP+24],EAX      Overwrite NUM1 in stack with sum.
          POP    ECX               Restore registers.
          POP    EBX
          POP    EAX
          POP    EDI
          RET                      Return.
```


Potprogrami

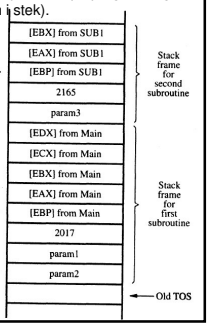
Sledeći primer predstavlja ilustraciju realizovanja potprograma, gde su parametri preneseni preko steka (sadržaj steka).



Potprogrami

Sledeći primer predstavlja ilustraciju realizovanja ugnježdenih potprograma, gde su parametri preneseni preko steka (glavni program i stek).

```
                  :
2000      PUSH    PARAM2      Place parameters
2006      PUSH    PARAM1        on stack.
2012      CALL    SUB1
2017      POP     RESULT      Store result.
          ADD     ESP,4       Restore stack level.
                  :
```



Potprogrami

Sledeći primer predstavlja ilustraciju realizovanja ugnježdenih potprograma, gde su parametri preneseni preko steka (potprogram 1).

```
2100  SUB1:  PUSH   EBP          Save frame pointer register.
             MOV    EBP,ESP      Load frame pointer.
             PUSH   EAX          Save registers.
             PUSH   EBX
             PUSH   ECX
             PUSH   EDX
             MOV    EAX,[EBP+8]  Get first parameter.
             MOV    EBX,[EBP+12] Get second parameter.
                    :
             PUSH   PARAM3       Place parameter on stack.
2160         CALL   SUB2
2165         POP    ECX          Pop SUB2 result into ECX.
                    :
             MOV    [EBP+8],EDX  Place answer on stack.
             POP    EDX          Restore registers.
             POP    ECX
             POP    EBX
             POP    EAX
             POP    EBP          Restore frame pointer register.
             RET                 Return to Main program.
```


Potprogrami

Sledeći primer predstavlja ilustraciju realizovanja ugnježdenih potprograma, gde su parametri preneseni preko steka (potprogram 2).

```
3000  SUB2:  PUSH   EBP          Save frame pointer register.
             MOV    EBP,ESP      Load frame pointer.
             PUSH   EAX          Save registers.
             PUSH   EBX
             MOV    EAX,[EBP+8]  Get parameter.
                    :
             MOV    [EBP+8],EBX  Place SUB2 result on stack.
             POP    EBX          Restore registers.
             POP    EAX
             POP    EBP          Restore frame pointer register.
             RET                 Return to first subroutine.
```


Ostale instrukcije

Instrukcije množenja i deljenja

U opštem slučaju, množenjem dva 32-bitna cela broja dobija se proizvod dužine 64 bita. Ipak, u mnogim aplikacijama, očekuje se da je rezultat 32-bitni binarni broj. Drugi slučaj se realizuje instrukcijom sledećeg oblika:

IMUL reg, src

gde reg predstavlja registar opšte namene.
Prva alternativa se realizuje instrukcijom:

IMUL src

i tada registar EAX predstavlja odredište. U slučaju da je proizvod dvostruke dužine, tada se viši bitovi rezultata smeštaju u EDX, a niži bitovi u EAX. Izvorni operand instrukcije može biti registar ili memorija.

Instrukcija deljenja ima sledeći oblik:

IDIV src

Izvorni operand je delilac. Deljenik je u registru EAX. Po izvršenju instrukcije, rezultat se smešta u EAX, a ostatak deljenja u EDX. Pre izvršenja deljenja, deljenik u EAX-u treba biti proširen znakom preko celog EDX-a. To se postiže izvršenjem instrukcije CDQ:

CDQ

Ostale instrukcije

MMX instrukcije

Dvodimenzionalna grafika ili video slika može biti predstavljena nizom velikih brojeva. Boja i osvetljenje svake tačke može da se enkodira u 8-bitni podatak. Procesiranje takvih podataka ima dve važne karakteristike:

- Manipulacija sa pojedinačnim pikselima često uključuje veoma proste aritmetičke ili logičke operacije.
- Za neke aplikacije koje zahtevaju prikaz u realnom vremenu, zahteva se veoma velika brzina izračunavanja.

Iste ove karakteristike se odnose i na sampliranje zvučnih signala i na procesiranje govora, a tu sekvenca označenih brojeva predstavlja uzorke kontinualnog analognog signala koji se uzimaju u određenim vremenskim intervalima.

U ovakvim aplikacijama, brzina procesiranja se postiže pakovanjem gore opisanih podataka u male grupe, gde se podaci u grupama mogu paralelno procesirati.

Skup instrukcija za IA-32 sadrži instrukcije koje paralelno rade nad podacima koji su spakovani u 64-bitne četvorostruke reči. Takve instrukcije se nazivaju MMX instrukcije – instrukcije za proširenje multimedije. Operandi ovih instrukcija mogu biti u memoriji ili u nekom od osam registara za rad sa brojevima u pokretnom zarezu i tada se na njih referiše sa MM0 do MM7.

Ostale instrukcije

Obezbeđene su instrukcije za prenos podataka između memorije i MMX registara. Instrukcija

PADDB MMi, src

sabira odgovarajuće bajtove dva osmobajtna operanda i smešta osam suma bajtova u odredišni registar. Izvorni operand može biti bilo memorija bilo MMX registar, a odredište mora biti registar.

Slične instrukcije postoje za operacije oduzimanja i za logičke operacije.

Česta operacija kod procesiranja signala je množenje kratke sekvence ulaznih signala sa tzv. težinskim konstantama i sabiranje tako dobijenih proizvoda, kako bi se oformio izlazni uzorak signala.

Kao što se i očekuje, postoji i MMX instrukcija koja kombinuje operacije sabiranja i množenja, a koja se izvršava nad 64-bitnim MMX operandom koji sadrži četiri 16-bitna podatka-uzorka.

Ostale instrukcije

SIMD instrukcije

Obezbeđen je i skup instrukcija koje se koriste za izvršavanje operacija na malim grupama brojeva u pokretnom zarezu.

SIMD – *Single Instruction Multiply Data* instrukcije su korisne za vektorska i matrična izračunavanja pri numeričkim kalkulacijama.

U Intel-ovoj terminologiji, ove instrukcije se nazivaju protočne SIMD proširene (SSE – *Streamed SIMD Extension*) instrukcije.

SSE instrukcije operišu nad složenim podacima dužine 128 bitova, od kojih se svaki sastoji od četiri 32-bitna boja u pokretnom zarezu. Za čuvanje takvih operanada obezbeđeni su osam registara dužine 128 bitova – ti registri nisu prikazani na prvoj shemi.

Dve osnovne operacije u voj grupi su operacije sabiranja i množenja.

Primeri programa

```
        LEA     EBP,AVEC        EBP points to vector A.
        LEA     EBX,BVEC        EBX points to vector B.
        MOV     ECX,N           ECX is the loop counter.
        MOV     EAX,0           EAX accumulates the dot product.
        MOV     EDI,0           EDI is an index register.
LOOPSTART: MOV  EDX,[EBP+EDI*4] Compute the product
        IMUL    EDX,[EBX+EDI*4]   of next components.
        INC     EDI             Increment index.
        ADD     EAX,EDX         Add to previous sum.
        LOOP    LOOPSTART       Branch back if not done.
        MOV     DOTPROD,EAX     Store dot product in memory.
```

skalarni proizvod vektora

Primeri programa

```
for (j = n−1; j > 0; j = j − 1)
    { for ( k = j−1; k >= 0; k = k − 1 )
        { if (LIST[k] > LIST[j])
            { TEMP = LIST[k];
              LIST[k] = LIST[j];
              LIST[j] = TEMP;
            }
        }
    }
```

```
        LEA     EAX,LIST        Load list pointer base
        MOV     EDI,N             register (EAX), and initialize
        DEC     EDI               outer loop index register
                                  (EDI) to j = n − 1.
OUTER:  MOV     ECX,EDI         Initialize inner loop index
        DEC     ECX               register (ECX) to k = j − 1.
        MOV     DL,[EAX + EDI]  Load LIST(j) into register DL.
INNER:  CMP     [EAX + ECX],DL  Compare LIST(k) to LIST(j).
        JLE     NEXT            If LIST(k) ≤ LIST(j), go to
                                  next lower k index entry;
        XCHG    [EAX + ECX],DL  Otherwiser, interchange LIST(k)
                                  and LIST(j), leaving
        MOV     [EAX + EDI],DL    new LIST(j) in DL.
NEXT:   DEC     ECX             Decrement inner loop index k.
        JGE     INNER           Repeat or terminate inner loop.
        DEC     EDI             Decrement outer loop index j.
        JG      OUTER           Repeat or terminate outer loop.
```

sortiranje niza

Primeri programa

```
Subroutine
INSERTION:  MOV   RNEWID,[RNEWREC]
            CMP   RHEAD,0           Check if list empty.
            JG    HEAD
            MOV   RHEAD,RNEWREC     If yes, new record becom
            RET                       one-entry list.
HEAD:       CMP   RNEWID,[RHEAD]    Check if new record
                                      becomes head.
            JG    SEARCH
            MOV   [RNEWREC+4],RHEAD If yes, make new record
            MOV   RHEAD,RNEWREC       the head.
            RET
SEARCH:     MOV   RCURRENT,RHEAD    Otherwise, use
LOOPSTART:  MOV   RNEXT,[RCURRENT+4]  RCURRENT
            CMP   RNEXT,0             and RNEXT
            JE    TAIL                to move through
            CMP   RNEWID,[RNEXT]      the list to find
            JL    INSERT              the insertion point.
            MOV   RCURRENT,RNEXT
            JMP   LOOPSTART
INSERT:     MOV   [RNEWREC+4],RNEXT
TAIL:       MOV   [RCURRENT+4],RNEWREC
            RET
```

umetanje u listu

Primeri programa

Subroutine

```
DELETION:   CMP   RIDNUM,[RHEAD]        Check if head.
            JG    SEARCH
            MOV   RHEAD,[RHEAD+4]       If yes, remove.
            RET
SEARCH:     MOV   RCURRENT,RHEAD        Otherwise,
LOOPSTART:  MOV   RNEXT,[RCURRENT+4]      use RCURRENT
            CMP   RIDNUM,[RNEXT]          and RNEXT
            JE    DELETE                  to move through
            MOV   RCURRENT,RNEXT          the list to
            JMP   LOOPSTART               find the record.
DELETE:     MOV   RTEMP,[RNEXT+4]
            MOV   [RCURRENT+4],RTEMP
            RET
```

brisanje iz liste

Zadaci

-
-
-