Assemblerinis programavimas

Kurso medžiaga parengta moduliui FMITB05044 "Assemblerinis programavimas".
Jam yra skirta 3.5 kedito.
Medžiaga pagal mokymo programą skirta Inžinerinės informatikos studentams ir skirta personalinio kompiuterio vartotojams, kuriems reikia artimiau susipažinti su **Intel** kompanijos mikroprocesorių architektrinėmis savybėmis ir išmokti sudarinėti programas šiems kompiuteriams assemblerio kalboje.

Vertinant studentų žinias, atsiskaitymui studentai turi pateikti parašytą programą assemberio kalboje, mokėti paaiškinti ją. Kadangi programos komandos faktiškai atitinka mašinines komandas, studentas turi mokėti vykdyti programą derinimo režime, kad galėtų parodyti kaip keičiasi kintamųjų reikšmės operatyvinėje atmintyje ir procesoriaus registruose, vykdant programos komandas.

KOMPIUTERIO ARCHITEKTŪRA
Kas aprašo kompiuterio architektūrą:

- Tai abstraktus PK supratimas, kuris atspindi jo struktūrinę, schematinę ir loginę organizaciją.
- Tai struktūrinė PK schema
- Priėjimo prie struktūrinės schemos elementų priemonės ir būdai
- Interfeisų organizacija ir jų laipsniškumas
- Registrų rinkinys ir jų panaudojimas
- Atminties organizavimas ir adresavimo būdai
- Duomenų formatai ir jų atvaizdavimo būdai
- Mašininių komandų rinkinys
- Mašininių komandų formatai
- Pertraukimų apdorojimas

Kompiuterio architektūros supratimas programuotojui duoda visą būtiną informaciją apie kompiuterį. Visi kompiuteriai turi tam tikras bendras ir taip pat individualias architektūros savybes. Bendros architektūrinės savybės apsprendžiamos tuom, kad dauguma kompiuterių yra 4-5 klasės. Tos bendros savybės yra:

- Saugomos programos principas. Tai reiškia, kad programa ir jos duomenys yra vienoje OA adresinėje erdvėje.
- Mikroprogramavimo principas. Procesoriaus sudėtyje yra mikroprograminis valdymo blokas. Toks blokas kiekvienai komandai turi rinkinį signalų, kuriuos reikia sugeneruoti, kad atlikti mašininę komandą.
- Ištisinė atminties erdvė - tai OA ląstelių rinkinys, kurioms nuosekliai yra priskiriami numeriai
- Programos vykdymo nuoseklumas. Procesorius išrenka iš atminties komandas griežtai nuosekliai. Nuoseklumui pakeisti galima panaudoti specialias komandas.

Iš procesoriaus pusės nėra principinio skirtumo tarp komandų ir duomenų. Duomenys ir mašininės komandos užima tą pačią OA erdvę ir yra nuliukų ir vienetukų rinkinys. Procesorius išrinktą informaciją visada stengiasi traktuoti kaip komandą, o jei yra ne taip

tai įvyksta avarinis programos darbo pabaigimas. Todėl programoje svarbu tiksliai atskirti duomenis nuo komandų.

- Visiškai abejingas kompiuteris duomenims. Jam visiškai nesvarbu kokią logiką saugo patys duomenys.

Operatyvinė atmintis RAM

Tai lastelių rinkinys, naudojamas centrinio procesoriaus vykdomų programų ir informacijos saugojimui. Kiekviena iš tų lastelių gali saugoti skaitmeninę reikšmę ir yra numeruojamos . Tai reiškia, kad jos viena nuo kitos skiriasi savo adresu. RAM savybė yra ta, kad išjungiant maitinimą, visi duomenys yra panaikinami. Dėl tos priežasties reikėjo sukurti įrenginį nuolatiniam informacijos saugojimui. Taip atsirado ROM.

Procesorius

Procesorius yra kompiuterio smegenys. Dar kitaipjis vadinamas centrinis procesorinis įrenginys . Yra įvairių procesorių. Jie skiriasi pagal savo greitaeigiškumą, šinoos ir registrų skirsniškumu, adresuojamos operatyvinės atminties kiekiu ir kitais parametrais.

Įvedimo išvedimo įrenginiai

Per atatinkamas sąsajas yra galimybė pateikti informaciją į OA arba išvesti iš jos. Įvedimo įrenginiai yra klaviatūra, skaneris, ir kiti. Išvedimo įrenginiai yra spausdintuvas, vaizduoklis, ir kiti.

Pagalbiniai atminties vienetai

Antrinė atmintis yra magnetiniai diskai, juostoas ir kiti. Informacija juose saugoma įvardintuose failuose.

OPERATYVINĖS ATMINTIES ORGANIZAVIMAS

Tai lastelių rinkinys į kurį galima įrašyti arba iš jos pasiimti informaciją. Lastelė (baitas) sudaryta iš 8 bitų. Bitai numeruojami nuo 0 iki 7 iš dešinės pusės, o baitai numeruojami iš kairės pusės.

Baitas – mažiausia OA vieta, kuriai galima priskirti adresą.

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0								1							

Kiekviename bite gali būti įrašytas skaičius 0 arba 1. Visa informacija saugoma dvejetainėje sistemoje.

0	0	0	1	0	0	1	1
1				3			

=13h=19

Žodis- du gretimai einantys baitai. Žodžio adresas yra kairiojo baito adresas. Žodžio dydis yra 16 bitų.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								1							

Dvigubas žodis yra du gretimai einantys žodžiai.

A	A+1	A+2	A+3

Klausimai savikontrolei

1. Koks yra pats mažiausias kompiuterio atminties vienetas? Pasirinkite teisingą atsakymąišpateiktų.
   a) Baitas
   b) Bitas
   c) Žodis

PROCESORIAUS REGISTRAI

Bet kokia vykdoma programa turi savo žinioje tam tikrus mikropracesoriaus resursus. Tų resursų rinkinys atspindi mikroprocesoriaus programinį modelį. Tai yra:

- Bendros paskirties registrai
- Registrai darbui su plaukiojančiu kableliu (kooprosesorius)
- Būsenos ir valdymo registras (vėlevėlių)
- Segmentiniai registrai
- Komandos nuorodos registras
- Sisteminiai registrai

32 skirsnių procesoriuose yra padidintos kompiuterio galimybės, bet žemesnieji registrai sutampa ir pagal pavadinimus ir pagal paskirtį su 16 skirsnių procesoriaus registrais. Yra 12 registrų, kurie dar kitaip gali būti skirstomi į tris grupes: duomenų registrus, registrus –nuorodas, segmentinius registrus. Jie visi vadinami bendro naudojimo registrais.

Duomenų registrai

Nnaudojami bet kokiam laikinam duomenų išsaugojimui. Su tais duomenimis galima atlikti bet kokius veiksmus. Tai yra registrai:

AX – AH+AL – akumuliatorius
BX – BH+BL – bazinis registras
CX – CH+CL – skaitliuko registras
DX – DH+DL – duomenu registras

Yra galimybė kreiptis į visą registrą ar į jo dalį. Kai kurios komandos savo vykdymui naudoja konkrečius registrus. Tai dalybos ir daugybos komandos.

Registrai – nuorodos

SI – šaltinio registras
DI – priėmėjo registras
BP – bazės nuoroda
SP – steko nuoroda

- SI ir DI yra ekvivalentūs vienas kitam, tačiau jų paskirtis yra saugoti duomenis tam tikros bazės atžvilgiu, išrenkant duomenis iš operatyvinės atminties arba juos urašant į operatyvinę atmintį.. Bazės adresas nurodomas registere BP arba BX.
- BP gali būti naudojamas kaip nuoroda dirbant su duomenimis steke, taip pat daugelyje aritmetinių ir loginių komandų laikinam duomenų saugojimui.
- SP naudojamas tik darbui su steku ir visada nurodo steko viršūnę.

Segmentiniai registrai

Jie saugo pradinius programos segmentų adresus ir leidžia kreiptis į juos
CS – komandų segmento registras
SS – steko segmento registras
DS – duomenų segmento registras
ES – papildomo duomenu segmento registras.

Kiti registrai

IP – šis registras vadinamas komandų nuoroda. Jis saugo komandos, kuri turi būti atlikta, adresą. Programiškai šis registras neprieinamas. Jo reikšmę keičia procesorius, atsižvelgdamas į vykdomos komandos ilgį.
FLAGS -veliavėlių registras, atspindintis procesoriaus būseną po komandos įvykdymo. Daugiausia gali būti 16 veliavėlių –kiekvienam bitui po vieną.Vėliavėlės būseną atspindi bito reikšmė.
Jei bito reikšmė 0- vėliavėlė neįjungta, o jei 1 tai vėliavėlė įjungta.

- 0 bitas-CF – pernešimo veliavėlė. Ji įsijungia, jei yra duomenų pernešimas (ją įtakoja aritmetinės operacijos, arba jei yra klaida, kreipiantis į sistemines funkcijas).
- 2 bitas–PF – pariteto veliavėlė. Įsijungia, jei 8 skirsnių rezultate yra lyginis vienetukų skaičius.
- 4 bitas–AF – papildomo pernešimo veliavėlė. Įsijungia, jei dirbame su supakuotais duomenimis.
- 6 bitas–ZF – nulio veliavėlė. Įsijungia, jei po operacijos įvykdymo rezultatas yra 0.
- 7 bitas–SF – ženklo vėliavėlė. Įsijungia, jei rezultato ženklas yra neigiamas.
- 8 bitas–TF – naudojama derinimo programose, kai jas norime atlikti žingsniais. Jei ji įjungta, procesorius po kiekvienos komandos įvykdymo leidžia sustabdyti porgramos vykdymą, norint analizuoti programos darbą.
- 9 bitas–IF – leidžia arba draudžia procesoriui reaguoti į pertraukimus nuo išorinių įrenginių.
- 10 bitas–DF – naudojama programose, dirbančioms su eilutemis, ir nurodo, kuria kryptimi turi būti peržiūrimi duomenys eiluteje.
- 11 bitas–OF – įsijungia tuo atveju, jei yra perpildymas.

Klausimai

1. Kokia yra registrų CS,DS,SS ir Espaskirtis?
2. Kas tai yra registrai nuorodos?
3. Kas tai yra vėlevėlių registras?
4. Kokia yra registro IP reikšmė vykdant programas

5. Ar gali programuotojas keisti programiškai registro IP reikšmę? Pasirink atsakymą

a) Taip
b) Ne

KOMANDŲ FORMATAI

Registras – registras

Kiekviena komanda sudaryt iš dviejų dalių – opreaciojos kodo ir operandų. Pagal operandų pateikimą yra keletas komandos formatų. Nuo to priklauso komandos ilgis.

OK d w 11 reg1 reg2

Komanda užima du baitus, atlieka veiksmus su duomenimis, patalpintais registruose. Po operacijos rezltatas yra formuojamas registre:
Reg1:=reg1*reg2
Reg2:=reg2*reg1

OK – skaičius, kuris nusako, koks veiksmas turi būti atliktas. W nustato operandų dydį: w=1, kai operandas yra žodis, w=0, kai operandas yra baitas. d=0, kai rezultatas saugomas Reg2, d=1, kai rezultatas saugomas Reg1.
Atatinkamai iš lentelės matome kokios formuojamos reikšmės komandos laukuose, kai programose naudojame atatinkamus registrus
Reg w=1 w=0 Reg w=1 w=0

Reg	W=1	W=0	Reg	W=1	W=0
000	AX	AL	100	SP	AH
001	CX	CL	101	BP	CH
010	DX	DL	110	SI	DH
011	BX	BL	111	DI	BH

Registras – atmintis

OK d w mod reg mem adr(0-2)
Komandos ilgis – nuo 2 iki 4 baitų.

Reg:=reg*adr
adr=adr*reg
Vėl **w** nustato operandų dydį: w=1, kai žodis, w=0, kai baitas. d=0, kai rezultatas saugomas atmintyje, d=1, kai rezultatas saugomas registre. **Mod** nusako, kiek baitų užima **operandas adresas** (jei mod=00, **adr** užima 0 baitų, jei mod=01, tai **adr** užima 1 baitą, jei mod=10, **adr** užima 2 baitus). **Mem** nurodo adresų modifikavimo būdą:

a8 adresas telpa 1 baite, a16 – adresas telpa dviejuose baituose. Jei komandoje adresas nenurodytas, jis skaitomas lygiu 0. jei mod=11, komanda pavirsta komanda registras – registras tipo.

Registras – tiesioginiai duomenys

Komanda užima nuo 3 iki 4 baitų.

OK S W 11 OK1 reg im(1-2)
OK1 – nurodo kokią komandą turim atlikti, rezultatas saugomas registre, w nurodo operando dydį. OK1 nurodo grupę komandų, kuriai priklauso ši komanda, OK patikslina komandą. Tiesioginiai duomenys **im** gali užimti 1 arba 2 baitus.

Atmintis – tiesioginiai duomenys

Komandos ilgis –nuo 3 iki – 6 baitų
OK S W 11 OK1 mem adr(0-2) im(1-2)
Adr:=adr*im
OK patikslina pačią komandą.

Klausmai

1. Nuo ko priklauso komandos ilgis? Pasirinkite atsakymą.
    a) Nuo duomenų iilgio
    b) Nuo duomenų adresavimo būdo
    c) Nuo komandos formato
    d) Dalinai nuo visų išvardintų priežasčių

2. Koks yra didžiausias irmažiausias komandos ilgis?

ASSEMBLERIO KALBOS PAGRINDAI

Programa – komandų rinkinys, kurios kompiuteryje gali atlikti nurodytus veiksmus. Assembleris – simbolinė programavimo kalba, leidžianti užrašyti programą. Assembleris beveik mašininio lygio kalba, atliekanti aukšto lygio veiksmus. Tai programavimo kalba, leidžianti užrašyti programą ir ją transliuoti į mašininį kodą.
Programa dažniausiai sudaryta iš trijų segmentų. Pirmas – duomenų aprašymo segmentas, antras- komandų aprašymo segmentas, trečias –steko segmentas. Duomenų aprašymo segmente tam tikrų operatorių pagalba aprašome duomenis, su kuriais dirbs programa. Komandų segmentas visada naudojamas aprašyti veiksmus, kuriuos turi atlikti programa. Komandų pagalba aprašome programos algoritmą, kurį turime realizuoti. Steko segmentas – pagalbinė atmintis, naudojama tarpinių rezultatų, adresų saugojimui. Kiekvienas segmentas prasideda specialiu operatoriumi SEGMENT ir baigiasi operatoriumi ENDS. Visi jie turi savo vardus. Vsi šie segmentai sudro programos išeities tekstą.

Sakiniai
Programos tekstas uzrašomas sakiniais, kuriais aprašoma komanda, operandas arba komentaras.Sakinio struktūra yra pateikta žemiau ir skiriasi dėl duomenų aprašymo segmento ir dėl komandų segmento.

[vardas] direktyva operandas1 [,operandas2…] [;komentaras] ;duomenų aprašymo segmentas
[žyme:] komanda operandas1 [,operandas2…] [;komentaras] ;komandų segmentas

[vardas] eina kartu su direktyva. Transliavimo metu, vardui gali būti priskirtos tam tikros charakteristikos. Vėliau vardas programoje gali būti panaudotas nurodant duomenų vietą.
[vardas] atskiria vieną direktyvą nuo kitos, tačiau direktyva gali būti ir be jo.
[žyme:]nurodo pirmo komandos baito, kuriai ji yra priskirta, adresą.
Komanda / direktyva yra kodai, kurie atitinka tam tikra mašininę komandą arba transliatorius iš tos direktyvos suformuoja atatinkamus duomenis.
Komandos ar /direktyvos operandai aprašo objektus, su kuriais atliekami veiksmai. Jais gali būti skaičiai, išraiškos, adresai, simboliniai adresai, registrų numeriai. Išraiškose gali būti skaičiai, išraiškos, adresai, simboliniai adresai, registrų numeriai (vardai). Išraiškose gali būti naudojami aritmetinių operacijų simboliai. Iš sakinių formuojamas programos išeities kodas ir atatinkamai paskirstomas tarp programos segmentų.

Sakiniams taikomos rašymo taisyklės:
- Negalima sakinio perkelti į kitą eilutę.
- Negalima vienoje eiluteje rašyti kelių sakinių.
- Sakinys ne ilgesnis nei 131 simbolis.

Komentarai
Jie gali būti rašomi bet kurioje programos vietoje ir prasideda kabliataškiu. Tuščia eilutė taip pat suprantama kaip komentaras.
Užrašant komandą, operacijos kodas yra būtinas atributas. Tai kodas, nurodantis koks veiksmas turi būti atliktas. Esant ne vienam operandui, vienas nuo kito jie skiriami kableliais. Bent vienas operandas komandoje yra būtinas.

Direktyvos
Direktyva yra skirta duomenų aprašymui programose. Jei ji turi vardą, vėliau šis vardas gali būti panaudotas komandose. Direktyvos vardas nurodo pirmojo aprašyto baito adresą segmento pradžios atžvilgiu. Direktyvos yra tarnybiniai žodžiai, kuriuos supranta transliatorius.

Direktyva DB aprašo duomenis, kurie operatyvineje atmintyje yra formuojami viename baite. Kintamojo tipas yra BYTE, o ilgis 1 baitas.

[vardas] DB operandai [;komentaras]
Operandai nurodo reikšmes, kurios formuojamos opreatyveneje atmintyje. Esant keletui operandų, jie nurodomi per kalbelį. Pirmajam iš jų pirskiriamas nurodytas

[vardas] - suteikiamas adresas. Yra 2 būdai formuoti baito reiksmę. Su "?" arba nurodant konkrečią reikšmę.
Operatyvineje atmintyje nuo to paties adreso gali prasideti skirtingo dydžio duomenys: baito, žodžio, dvigubo žodžio arba eilutė.

- Operandas kosntanta su reikšmėmis nuo –128 iki 255. Bet kuriuo atveju atmintyje išskiriamas 1b.
- Direktyva su keliais operandais- vienas iš būdų naudojamas aprašyti masyvą. Masyvuose vardas paparstai suteikiamas pirmam elementui Jei yra keli operandai, simboliais aparšomi operatyvineje atmintyje greta vienas kito, juos galima aprašyti kaip eilutę.

- Direktyva su keliais operandais pateikta žemiau.

X DB 10,15,'A' kur X-yra direktyvos vardas
Operatyvinėje atmintyje duomenys išsidėsto sekančiai:

10	15	A kodas

A raidė yra keičiama jos šešioliktainiu ASSCI kodu. Tą patį efektą gautume aprašydami

```
X   DB  10
    DB  15
    DB  'A'
```

- Pasikartojimo Operatorius **DUP**.

Jis naudojamas sutrumtinti duomenų aprašymą, kai reikia aprašryti keletą vienodų reikšmių greta viena kitos.

```
A   DB  5 DUP('*')
```

Operatyvinėje atmintyje susiformuos penki baitai su ta pačia reikšme '*'. Taip pat galima naudoti masyvų aprašymui.

```
A   DB  '*','*','*','*','*'    tas pats kas būtų aprašyta su pasikartojimo
operatoriumi
B   DB  10 DUP(10 DUP(?))
```

Simbolius galima užrašyti dvejopai:
nurodant juos tarp kabučių arba nurodant to simbolio skaitmeninį kodą iš ASSCI lentelės.

Direktyva DW

Galima aprašyti žodžio ilgio duomenis.

A DW ? bus išskirtas žodis. Kintamojo tipas WORD, ilgis 2 baitai

?	?

Į žodį galima įrašyti skaičių nuo –32768 iki 65535 .

A DW 1234 Atmintyje žodis saugomas sukeistam pavidale.

34	12

B DW '01'

31h	30h

Informacija saugoma šešioliktainėje sistemoje.

Direktyva DD
Galma formuoti duomenis dvigubame žodyje. Kintamojo tipas DWORD, kintamojo ilgis 4 baitai. Į dvigubą žodį galima įrašyti skaičių nuo -2 ^{31} шлш 2 ^{31}-1.

Konstantas galima užrašyti ir su direktyva EQU.

Direktyva EQU
[vardas] EQU operandas
Su EQU galima užrašyti tik vieną konstantą,t.y. galime parašyti tik vieną operandą.
Star EQU ‘*‘
T DB ‘*‘ atitiktų Star EQU ‘*‘
Su direktyva EQU sufrmuotų duomenų pakeisti negalima. Jie formuojasi tarp segmentų ir nepriskiriam jokiam segmentui.

Operandas-vardas
EQU pagalba galima formuoti vardų sinonimus
SUM EQU AX. Registrui AX yra suformuojamas vardas SUM. Vėliau programoje į registrą AX galima kreitis kaip į kintamąjį SUM, ir tai yra aiškiau.

Operandas-konstantinė išraiška
N EQU 100 N yra konstantos vardas
K EQU N*2-1 atitiktų reikšmę 199
X DB N DUP(?) būtų aprašytas masyvas X DB 100 DUP(?)

Su EQU galima aprašyti bet kokį tekstą

S EQU “Sveiki”

Išraiškos
Jos naudojamos patogiai užrašyti direktyvų ir komandų operandus.

Konstantinės išraiškos
Jų rezultatas yra sveiki skaičiai. Paprasčiausios išraiškos yra skaičiai, simboliai, eilutė iš dviejų simbolių, konstantos vardas.

Adresinė išraiška
Ji visada užima 16 bitų. Jos reikšmė yra adresas. Tai kintamojo vardas, aprašytas su direktyvomis DB,DW,DD, arba žymės vardas, arba skaitliukas.
C DB ?
D DW C
Į atmintį bus užrašomas C adresas. Adresinė išraiška formuoja absoliutinį adresą. Adresinė išraiška gali būti naudojama, kai reikia kreiptis į lasteles,kurios nėra įvardintos. Transliavimo metu sakiniai peržiūrimi vienas paskui kitą ir nustatomos reikšmės yra

formuojamos OA viena paskui kitą, kartu įsimenant ir jų adresus. Be to, pagal aprašymą, assembleris įsimena ir kiek baitų užima kintamasis. Tai vadinamas kintamojo tipas. Reikšmė ir tipas aprašo lastelę su nurodytu vardu. Nuo to paties adreso OA gali prasidėti skirtingo tipo duomenys- baito, žodžio,dvigubo žodžio. Kad galėtume dirbti su skirtingo tipo duomenimis, naudojamas

duomenų tranformavimo operatorius.

<tipas> PTR <išraiška>

Tipas gali būti BYTE, WORD, DWORD
pvz.: MOV BYTE PTR [SI], 0
Į atminties vietą pagal adresą , esantį registre SI, nusiūsime baito ilgio nulį.
Kadngi nuo tos vietos gali būti baitas ar žodis ar netgi eilutė, tai reikia patikslinti su kokio dydžio duomenimis mes noime dirbti.

Klausimai

1. Kas tai yra direktyva?
2. Kuos skiriasi duomenys aprašyti su direktyva DB ir ir EQU?
3. Nurodykite koks ilgis baitais operatvinėje atmintyje formuojamas su skirtingo tipo direktyvomis.
4. Kas tai ra duomenų transformaimo operatorius?

SEGMENTINIS ADRESAVIMAS

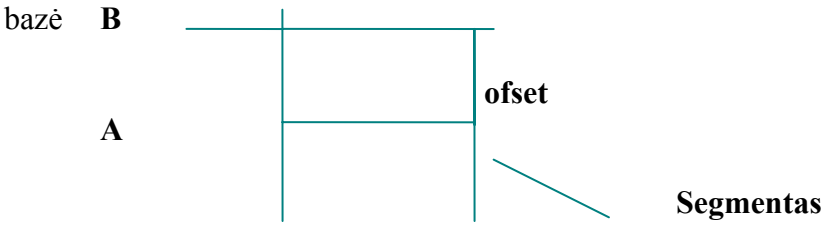
Pagrindinė procesoriaus charakteristika – registrų skirsniškumas (bitų skaičius registre), taip pat išorinių adresų ir duomenų šinų(linijų) kiekis. Maksimalus sveikas skaičius, su kuriuo gali dirbti mikroprocesorius – 2^{16} -1=65kb.
Adresinė atmintis turi 20 linijų (2^{20} - 1=1Mb). Tam kad su 16 skirsnių adresais būtų galima kreiptis į bet kurį 20 skirsnių atminties tašką, procesoriuje numatytas segmentinis atminties adresavimas, kuris realizuotas keturių segmentinių registrų pagalba. Segmento adreso esmė yra ta, kad 20 skirsnių adresas išskaičiuojamas sudedant 20 skirsnių segmento pradžios adresą, kuriame yra minėta lastelė su 16 skirsnių poslinkiu nuo segmento pradžios.
Pradinis segmento adresas be 4 žemiausių bitų yra saugomas viename iš segmento registrų. Ši reikšme vadinama segmentiniu adresu. Kiekvieną kartą į vidinį registrą užkraunant segmento adresą, procesorius automatiškai jį padalija iš16 ir tokiu būdu bazinis adresas saugomas viename iš vidinių registrų. Esant reikalui kreiptis į tam tikrą lastelę, procesorius prideda prie bazinio adreso lastelės poslinkį nuo segmento pradžios ir taip gaunamas fizinis lasteles adresas.
Daugyba iš 16 padidina adresuojamų lastelių diapazoną iki 1Mb. Taigi kreipinys į bet kurią lastelę vykdomas tik per segmentus, t.y. logiškas erdves uždedamas ant reikiamų fizinių erdvių operatyvineje atmintyje. Segmento dydis negali buti didesnis nei 64 Kb.
Pradinis segmento adresas yra dalintinas iš 16, t.y. be jaunesniojo 16-nio skirsnio

užrašomas į vieną iš segmentinių registrų. Adresuojamos lastelės poslinkis, kreipiantis į ją per kažkokią komandą, nurodomas naudojant tam tikrus adresavimo būdus. Visi segmentai operatyvineje atmintyje yra išdėstomi tokiu pat būdu, kaip jie aprašyti programos pradiniame tekste. Prieš komandų segmentą sistema patalpina papildomą (sisteminį) segmentą, t.y.programos prefixą, kur patalpinta sistemine informacija, reikalinga programos vykdymui. Užkraunant programą į operatyvinę atmintį abu, ES ir DS registrai, iguana ta pačią reikšmę ir rodo į programos prefixo pradžia. Tai reiškia, kad iš programos yra galimybe kreiptis į tą papildomą sritį operatyvineje atmintyje. Komandų segmento adresas patalpinamas į CS ir dažniausiai registro IP reikšmė tampa lygi 0, nes programa pradeda darbą nuo 1 komandos segmente. IP pradinė reikšmė formuojama iš programos žymės, kuri pateikiama paskutiniame programos sakinyje. Registro SS reikšmė formuojama sistemai uzkraunant programą į atmintį ir lygiagrečiai formuojama registro SP reikšmė, kuris yra nuoroda į steko viršunę. SP reikšmė priklauso nuo steko dydžio ir visada lygi steko dydžiui programos užkrovimo metu. Visi programos segmentai prasideda nuo paragrafo ribos (adresai dalinami iš16). Kadangi segmento ilgis dažnai nėra kartotinas 16, todėl tarp segmentų operatyvineje atmintyje atsiranda tarpai. Paprastai paskutinė komandos adresas **INT 21h** (grįžimas į DOS) ir programos ilgis nėra kartotinas 16, tas tarpas iki sekančio segmento užpildomas 0, kurie deasembliuojant virsta komanda Add.

Adresų segmentavimas

PK priskiriamas prie skaičiavimo mašinų, kur dėl komandų sutrumpinimo naudojamas bazinis atminties adresavimas. Kas tai yra?

Jei OA yra 2^k lastelių, tai nuorodai į tas lasteles reikalingi k- skirsnių adresai. Tokius adresus vadiname absoliutiniais arba kitaip tariant fiziniais adresais. Kuo didesnė atmintis tuo daugiau skirsnių reikia adresui nurodyti. Tai iš karto ilgina komandą ir tuo pačiu ilgina programą. Reikia sutrumpinti komandos ilgį, kad sutrumpinti programą. Tam tikslui atmintis dalinama į gabalus, kurie vadinami segmentais. Pradiniai segmentų adresai gali būti bet kokie, tačiau yra apribojamas segmento ilgis. Segmento ilgis negali būti didesnis negu 2^m baitų, kur m<k. Tomis sąlygomis absoliutinį bet kurios lastelės adresą A galima pateikti kaip sumą **A=B+ofset**, kur **B** segmento bazė, kuriam priklauso lastelė **A**, o **ofset** yra lastelės poslinkis nuo bazės pradžios arba sąlyginis adresas išskaičiuotas nuo segmento pradžios t.y. nuo adreso **B**.



Segmento dydžio apribojimas reiškia, kad **0<=ofset<=2^m -1**, ir todėl poslinkiui užrašyti užtenka **m** skirsnių. Vadinasi sumoje **A=B+ofset** didžioji adreso dalis tenka bazei **B**, o **ofset** tai tik nedidelė adreso dalis. Atsižvelgiant į tai elgiamės sekančiai:

jei komandoje reikia nurodyti absoliutinį adresą **A**, tai didžioji dalis sudietės t.y. **B** "paslepiama" tam tikrame registre R, o komandoje nurodomas tik tas registras ir mažoji dalis **ofset**, t.y. vietoj komandos **OK … A** .. naudojame komandą **OK….R ofset**….

Ką tas duoda?
Kadangi komanda dirba su vykdomuoju adresu, o jis išskaičiuojamas pagal formulę **A=[R]+ofset=B+ofset**, tai komanda dirbs su reikiamu adresu **A**. Iš kitos pusės laukai **R** ir **ofset** užima mažai vietos komandoje, todėl komandos dydis sumažėja palyginus su tuo atveju, kai komandoje nurodomas absoliutinis adresas. Pakeliui pakomentuosime, kad vieną kartą užrašius į registrą **R** segmento bazę, toliau mes galime nekeisdami to registro reikšmės, panaudoti jį kreipiantis į bet kurią šio segmento lastelę. Jei programos duomenys patalpinti atmintyje kompaktiškai, tai persiuntimų į registrą bus nedaug. Registrai, kuriuose saugomas bazės adresas, vadinami baziniais registrais, o užrašymas į komandą ne absoliutinių adresų, o bazinių adresų ir poslinkio, vadinamas baziniu adresavimu.

Adresų segmentavimo ypatumai
Pirmiausia pastebėsime, kad PK vietoj termino bazinis adresavimas naudojamas adresų segmentavimas, o baziniai registrai vadinami segmentiniais registrais. Be to, atminties dydis PK lygus 1 Mb, t.y 2^{20} baitų (k=20) , todėl absoliutiniai adresai čia užima 20 skirsnių , o segmento dydžiai neturi viršyti 64 Kb. t.y. 2^{16} baitų (m=16), todėl poslinkis čia yra 16 skirsnių adresai.
Pirmoji segmentinių adresų ypatybė yra ta, kad segmentiniais registrais gali būti ne bet kokie registrai, bet tik CS, DS, SS, ES. Žinoma pradiniai segmentų adresai gali būti saugomi ir kituose registruose, tačiau naudoti tuos kitus registrus segmentavimui negalima.
Kadangi PK viso tik keturi segmentiniai registrai, tai vienu metu dirbti galima su keturiais segmentais atmintyje. Nustatome kiekvieno segmento pradžiai savą registrą ir toliau visas to segmento lasteles segmentuojame pagal jį. Tačiau jei reikia daugiau segmentų, tuomet kai reikia kreiptis į pvz. penkto segmento lasteles, pasirenkame vieną iš segmentinių registrų ir kur nors įsimename jo turinį. O po to į šį registrą įrašome penktojo segmento pradžią ir toliau naudojame tą registrą priėjimui prie penktojo segmento lastelių. Vėliau, jei reikia, galima atstatyti ankstesnę to registro reikšmę, kad jis vėl rodytų į vieną iš 4 segmentų. Tokiu būdu, keisdami segmentinių registrų reikšmę, galime dirbti su bet kokiu segmentų skaičiumi atmintyje.
Pastebėsime, kad segmentų tarpusavio išsidėstymas atmintyje gali būti bet koks. Jie gali nepersikloti, dalinai persikloti ar net pilnai sutapti, t.y. segmentiniai registrai gali rodyti į to paties segmento pradžią. Segmentų dydis nustatomas programos autoriaus, tik svarbu kad jų dydis neviršytų 64 Kb. Nustačius segmentinį registrą segmento pradžiai, potencialiai galima segmentuoti 64 kb, o kiek realiai bus išnaudota, t.y koks bus segmento dydis, jau nustato pats programos autorius.
Antroji adresų segmentavimo savybė yra, kad užrašant komandą asemblerio kalboje, kreipinys į segmentinį registrą nurodomas per standartinę konstrukciją:

<segmentinis registras>:<adresinė išraiška>, kuri vadinama adresine pora, kuri sako, kad adresas adresinėje išraiškoje turi būti segmentuotas pagal nurodytą registrą.

Pvz. komandoje MOV AX,ES:X kintamojo X adresas bus segmentuojamas, naudojant ES registrą. Užrašus CS:, DS:, SS: ar ES: priimta vadinti segmentinio registro prefiksais. Asemblerio kalboje prefiksas visada užrašomas prieš adresą, kuris turi būti segmentuotas. Tačiau mašininėj kalboje situacija šiek tiek skiriasi: čia prefiksas statomas prieš visą komandą. Pvz. komanda MOV AX,ES:X iš tikrųjų užrašoma dviejų mašininių komandų pagalba:
ES:
MOV AX,X
ES: - tai speciali komanda (be operandų), kuri ir vadinama prefiksu. Viso tokių komandų-prefiksų yra keturios, po vieną kiekvienam segmentiniam registrui. Pati savaime ši komanda-prefiksas nieko nedaro, bet ji veikia sekančią už jos komandą: prefiksas sako, kad adresinis operandas sekančioj komandoj turi būti segmentuotas pagal atatinkamą registrą. Jei prefiksas pastatytas prieš komandą, kur nėra adresinio operando, tai jis suprantamas kaip tuščia komanda. Taigi, mašininėse komandose segmentiniai registrai nurodomi ne viduje komandos, o prieš ją. Tačiau asemblerio kalboje prefikso rašyti atskirai negalima , pvz. ES:MOV AX,X skaitoma klaidinga žiūrint iš asemblerio pusės. Prefiksas būtinai turi būti komandos viduje ir prieš adresą.
Trečia adresų segmentavimo ypatybe susijusi su segmentinių registrų dydžiu. Bendra adresų bazavimo schema supranta, kad bazinių registrų dydis pakankamai didelis ir juose gali būti patalpintas bet koks absoliutinis adresas, bet kokia bazė. Tačiau PK ši savybė nevykdoma. PK absoliutiniai adresai 20-skirsnių, o visi registrai, tame tarpe ir segmentiniai, 16- skirsnių. Natūralu, kyla klausimas : kaip 16 skirsnių segmentiniuose registruose sekasi patalpinti 20 skirsnių bazinius adresus? PK ši problema išspręsta sekančiu būdu. Iki šiol mes skaitėme, kad bazės adresas yra bet koks. PK pradiniams segmentų adresams taikomi apribojimai: kaip bazę galima naudoti bet kokį adresą, bet kartotinį 16. Šių adresų ypatybė ta, kad juose 4 paskutiniai bitai yra nuliai, arba ką reiškia šešioliktainėje šių adresų formoje paskutiniai skaičiai visada nuliai, t.y. jie turi XXXX0h formą, kur X- bet koks skaičius. Jei taip , tai tą nulinį skaičių galima tiesiogiai nenurodyti, o tik įsivaizduoti. Taip ir daroma: segmentiniuose registruose saugomi tik segmento pradinio adreso pirmieji 16 bitų, t.y. pirmieji keturi šešioliktainiai skaitmenys. Pvz. Jei segmento pradžia yra adresas 12340h, tai segmentiniame registre bus saugoma 1234h. Pradinis segmento adresas be paskutinio šešioliktainio 0 vadinamas segmento numeriu ir žymimas kaip seg. Procesorius, natūralu, įvertina šią segmentinių registrų savybę ir segmentuojant pirmiausia prie segmentinio registro prirašo iš dešinės pusės tą nenurodytą nulį ir tik tada prideda poslinkį, nurodytą komandoje. Sąlyginai tai galima atvaizduoti

CP=xxxx		xxxx0	bazė
OK….CP:YYYY	==	YYYY	poslinkis
		ZZZZZ	absoliutinis adresas

Kadangi nulio prirašymas iš dešinės atitinka daugybai iš 16, tai absoliutinio adreso formulę pagal adresinę porą CP:ofset galima išreikšti taip:

Aabs=16*[CP]+ofset

Pvz, jeigu ES=1234h, tai adresinė pora ES:53h užduoda absoliutinį adresą 16*1234h+53h=12340h+53h=12387h.
Dar patikslinsim. Kaip žinia, adresus komandoje galima modifikuoti per registrus BX, BP, SI, DI. Pirmiausia atliekama adreso modifikacija per registrus-modifikatorius, k rezultate gauname adresą, kurį vadiname vykdomuoju. Tuo pat metu primename, kad skaičiavimas vyksta pagal modulį 2^{16}, t.y. vykdomasis adresas visada 16 skirsnių adresas. Po to adresas suprantamas kaip poslinkis ir būtent jis segmentuojamas, t.y. prie jo pridedamas segmentinio registro turinys padaugintas iš 16.Toks sumavimas atliekamas pagal 2^{16} modulį, kad neviršyti maksimalaus galimo adreso. Tokiu būdu tikslesnė formulė išskaičiuojamam absoliutiniam adresui yra Aabs=(Avykd+16*[CP]) mod $2^{20.}$
Taigi jei laikytis nuostatos apie segmentinius registrus, tai programų tekstas asemblerio kalboje bus trumpesnis ir mašininės programos ilgis bus mažesnis. Jei tokios nuostatos nebus laikomasi, tai reikės komandose rašyti adresus pilnai per adresines poras.
Pvz., jei duomenų segmento pradžiai naudosime ne registrą DS, o SS, tai norint į X nusiūsti nulį reiks rašyti komandą MOV SS:X,0, nes negalima bus rašyti komandos MOV X,0, nes ji bus suprantama kaip MOV DS:X,0.
Yra vienas atvejis, kurio negalima pažeisti niekada, tai kad registras CS, visada turi būti naudojamas komandų segmento pradžiai nustatyti. Tai aiškinama tuo, kad PK komandos adresas, kuri turi būti vykdoma, visada nurodoma CS ir IP pora, kur komandų nuorodos registras IP saugo tos komandos poslinkį nuo komandų segmento pradžios. Todėl jei registras CS rodys ne komandų segmento pradžią, tai kompiuterio elgesys bus neprognozuojamas. Dėl tos priežasties prie perėjimo adresų negalima rašyti prefiksų skirtingų nuo CS.

Yra du atvejai kai mes turime tiksliai nurodyti segmentinį registrą.
Pirmas atvejis kai mes, prisilaikydami visų susitarimų, tam tikrais atvejais norime turėti išimtį.
Pvz. Jei į registrą AX norime pakrauti lastelės L turinį, kai ji yra komandų segmente. Tokiu atveju MOV AX,L netinka, nes ji suprantama kaip MOV AX,DS:L ir į registrą bus patalpinta visai kitos lastelės reikšmė. Todėl mums reikia nurodyti segmentinį registrą, kuriame yra komandinio segmento pradžios adresas t.y. registrą CS. Todėl komandą reikia rašyti MOV AX,CS:L
Antras atvejis kai mums reikia dirbti su atminties segmentu, kuris nėra nei komandų, nei duomenų nei steko segmentas. Tuo atveju segmento pradžią nustatome per registrą ES ir kreipiantis į to segmento lastelę naudojame būtent tą registrą. Pvz. INC ES:Y. Čia prefikso ES praleisti negalima, nes jis čia nesuprantamas pagal nutylėjimą. ES registras skirtas darbui su papildomais atminties segmentais. Taigi yra atvejų, kai reikia tiksliai nurodyti segmentinius registrus, bet tai yra reti atvejai. Daugumoje komandų reikia nurodyti ne pilną adresinę porą o tik patį poslinkį.

Programiniai Segmentai

Segmentinių registrų naudojimas verčia mus organizuoti programą sekančiu būdu: viename segmente reikia talpinti visas programos komandas ir šio segmento pradžios adresą patalpinti į registrą CS. Kitame segmente reikia talpinti visus programai reikalingus duomenis ir šio segmento pradžios adresą reikia patalpinti registre DS, o trečias segmentas skiriamas stekui ir jo pradžios adresas talpinamas registre SS. Tačiau mūsų sakinius asemblerio kalboje atmintyje nustatome ne mes o pats assembleris. Kaip assembleris nustato kokius sakinius kokiame segmente talpinti? Apie tai mes turime informuoti jį per segmentus. Mes nustatome kokius sakinius kokiame segmente talpinti ir segmentas asembleryje aprašomas pagal tam tikrą struktūrą:

<Segmento vardas> SEGMENT <parametrai>
sakinys 1
sakinys n
<Segmento vardas> ENDS

Programiniam segmentui suteikiamas vardas, kuris turi būti pakartotas du kartus – direktyvoje SEGMENT, kuri atidaro segmentą ir direktyvoje ENDS, kuri uždaro segmentą. Tarp tų dviejų direktyvų gali būti nurodytas bet koks sakinių kiekis. Tos konstrukcijos esmė yra ta, kad mes pranešame assembleriui, kad visi sakiniai tarp šių direktyvų turi būti patalpinti viename atminties segmente. Bet tuo pačiu reikia prisiminti, kad segmento dydis neturi būti didesnis nei 64 kb, todėl sakinių turi būti tiek, kad jų bendras ilgis užimtų ne daugiau nei 64 kb, kitaip bus fiksuojama klaida. Pastebėsime, jog programoje gali būti keletas segmentų su tuo pačiu vardu. Skaitoma, kad tas pats segmentas dėl tam tikrų priežasčių yra aprašomas dalimis. Assembleris visus tuos sakinius apjungs į vieną visumą.
Parametrai SEGMENT direktyvoje reikalingi tada, kai programa didelė ir saugoma keliuose failuose, ir parametrai nurodo kaip tuos failus apjungti. Jei programa nedidelė ir telpa viename faile tai parametrai nereikalingi.

```
 A SEGMENT
A1 db 400h  DUP(?)
A2 dw 8
A ENDS

B SEGMENT
B1 dw A2
B2 dd A2
B ENDS

C SEGMENT
ASSUME ES: A,DS:B,CS:C
L: MOV AX,A2
MOV BX,B2
```

…
C ENDS

Direktyva ASSUME

Jei mes apjungiame sakinius į segmentą, tai assembleris patalpins juos viename atminties segmente. Tai reiškia, kad visas to segmento lasteles galima segmentuoti per tą patį segmentinį registrą. Koks tai registras? Tai nustato programos autorius. Tarkime mūsų pavyzdyje galime segmentui A pasirinkti registrą ES, o segmentui B registrą DS. Apie savo pasirinkimą mes pranešame assembleriui, kad jis galėtų transliuoti komandas sutinkamai su mūsų pasirinkimu. Esmė ta, kad jei komandoje prieš operandą yra nurodytas prefiksas, tai su tokiu prefiksu assembleris suformuos atatinkamą mašininę komandą. Tačiau rašyti kaskart prefiksą nemalonu, norėtūsi kad assembleris už mus tai padarytų pats. Todėl, kad assembleris galėtų tai padaryti, jis turi žinoti koks segmentinis registras yra pasirinktas konkrečiam segmentui. Informacija apie segmentų ir segmentinių registrų atitikimą pranešama per direktyvą ASSUME.

ASSUME <adresinė pora>{,<adresinė pora>{,}
Arba **ASSUME NOTHING,** kur **<adresinė pora>** tai
<segmentinis registras>:<segmento vardas> arba
<segmentinis registras>:NOTHING.
Taip mūsų pavyzdyje su direktyva ASSUME ES:A,DS:B,CS:C mes pranešame, kad segmentui A mes pasirinkome registrą ES, B-DS, o C-CS. Reikalaujam iš assemblerio, kad komandose, kur prefiksas nenurodytas, jis visus vardus iš segmento A transliuotų su prefiksu ES, o iš segmento B su prefiksu DS, o iš C su CS atatinkamai. Tokiu būdu komanda MOV AX,A2 bus suprantama kaip MOV AX,ES:A2, nes A2 yra segmente A, o jam pagal direktyvą ASSUME nurodytas registras ES. Komanda MOV AX,B2 bus suprantama kaip MOV AX,DS:B2 dėl tos pačios priežasties.
Taigi ASSUME direktyvos dėka mes pranešame assembleriui apie atitikimą tarp segmentinių registrų ir segmentų, ir įgaunam teisę nenurodyti komandose prefikso. Praleisti prefiksai bus keičiami savarankiškai.

ASSUME ypatybės

Pirmiausia pastebėsime, kad direktyva ASSUME neužkrauna į segmentinius registrus segmentų pradžios adresų. Šios direktyvos pagalba programuotojas tik praneša, kad bus toks segmentavimas ir adresų užkrovimas.
Direktyvą ASSUME galima patalpinti bet kurioje programos vietoje, bet paprastai tai daroma komandų segmento pradžioje. Esmė ta, kad informacija iš tos direktyvos reikalinga tik komandų transliavimo metu, todėl iki komandų segmento ši informacija nereikalinga, o pirmos komandos transliasvimo metu, kur nurodomas koks nors vardas, ši informacija jau reikalinga. Todėl komandų segmento pradžia kaip tik ta vieta kur yra prasmė patalpinti šią direktyvą. Direktyvoje būtinai turi būti nurodytas ryšys tarp registro CS ir to segmento, kitaip pirmos žymės atveju assembleris iš kart fiksuos klaidą.
Jei direktyvoje ASSUME nurodytos kelios adresinės poros su tuo pačiu segmentiniu registru, tai paskutinė iš tų porų atšaukia prieš tai einančią, kadangi kiekvienam

segmentiniam registrui galima priskirti tik vieną segmentą. Pvz ASSUME ES:A,ES:B assembleris supras kad registras ES nurodo į segmentą B. Tuo pačiu į vieną ir tą patį segmentą gali būti nurodyti keli segmentiniai registrai.
Pvz. ASSUME ES:A,DS:A skaitosi, kad į segmentą A nurodo ir registras ES ir registras DS. Assembleris turi teisę rinktis prefiksą pats su kuriuo jis dirbs. Paprastai assembleris pasirenka tą prefiksą kuris yra numatytas pagal nutylėjimą.
Jei ASSUME direktyvoje vietoj antro elemento adresinėje poroje yra parašytas NOTHING (ASSUME ES:NOTHING), tai tas reiškia, kad nuo to momento segmentinis registras nenurodo į jokį segmentą, kad assembleris neturi naudoti šio registro ir kad programuotojas ima sau atsakomybę tiksliai nurodyti šį registrą kai reikia. Programoje ASSUME gali būti nurodyta kelis kartus.
ASSUME ES:a
ASSUME DS:B,CS:C
Toks užrašymas ekvivalentus ASSUME ES:A,DS:B,CS:C
Kai tarp direktyvų yra kitų komandų, ne visada galima apjungti tas direktyvas.

Segmentinių registrų pasirinkimas transliavimo metu.
Yra taisyklės, kurių laikosi assembleris komandų transliavimo metu, kaip turi būti parenkamas prefiksas prieš adresą komandoje. Jei operande nėra vardo pagal kurį assembleris galėtų nustatyti kokiame atminties segmente yra patalpintas operandas, tai assembleris išrenka tą segmentinį registrą, kuris toje komandoje numatytas pagal nutylėjimą. Pvz. Komandoje MOV AX,[BX] bus išrinktas prefiksas DS:, o komandoje MOV AX,[BP] prefiksas SS:.
Jeigu operande nurodytas kintamojo vardas arba žymė, tai assembleris žiūri ar buvo tas vardas jau aprašytas programoje. Jei ne, jei tai nuoroda pirmyn, tai assembleris skaito, kad tas vardas turi būti segmentuojamas pagal registrą, kuris čia komandoje suprantamas pagal nutylėjimą. Pvz. Jei X nuoroda pirmyn , tai komandoje INC X[SI] bus suprantamas prefiksas DS, o komandoje INC X[BP+1] prefiksas SS:

Pradinis segmentinių registrų užkrovimas
Direktyva ASSUME mes pranešame kad visi vardai iš tam tikrų programinių segmentų turi susigeneruoti pagal nurodytus segmentinius registrus ir reikalaujame, kad assembleris formuotų mašinines komandas su atatinkamais prefiksais. Tai dar nereiškia kad programa veiks teisingai. Tarkime pas mus pagal komandą MOV AX,A2 bus suformuota komanda MOV AX,ES:A2, bet jei registras ES nerodys segmento A pradžios, tai tokia komanda veiks neteisingai. Reikia, kad segmentiniuose registruose būtų teisingos reikšmės- atatinkamų segmentų numeriai atmintyje. Programos pradžioje ten nieko nėra. Todėl programos darbas turi prasidėti nuo komandų, kurios į segmentinius registrus užkrautų atatinkamų segmentų numerius atmintyje.
Taigi registras DS turi būti nustatytas į segmento B pradžią. Primename kad assembleryje segmento vardui priskiriama reikšmė – pirmi 16 bitų to segmento pradinio adreso, o tai ir yra tas, kas turi būti patalpinta į segmentinį registrą, todėl reikia atlikti priskyrimą DS:=B. Tačiau tai atlikti su komanda MOV DS,B negalima, nes segmento vardas, tai konstantinė išraiška, t.y. tiesioginis operandas, o su komanda MOV uždraustas persiuntimas

tiesioginių duomenų į segmentinį registrą. Todėl tokį persiuntimą reikia atlikti per kitą nesegmentinį registrą, pvz. AX.

```
MOV AX,B
MOV DS,AX   ;DS=B
```

Analogiškai užkraunamas ir registras ES. Kas liečia registrą CS, tai jo užkrauti nereikia. Tai dėl to, kad programos vykdymo pradžiai tas registras jau bus užpildytas ir rodys komandų segmento pradžią. Tokį užkrovimą atlieka operacinė sistema, prieš perduodama valdymą programai, todėl užkrauti CS pačioje programoje nereikia.
Registras SS naudojamas steko segmento nustatymui . Užkrauti šį registrą galima dvejopai. Pirma - jį galima užkrauti pačioje programoje- taip kaip tai daroma su registru DS arba ES. Antra - tokį užkrovimą galima pavesti operacinei sistemai. Dėl to direktyvoje SEGMENT, kuri atidaro steko segmentą, reikia nurodyti specialų parametrą STACK, pvz. S SEGMENT STACK. Tuo atveju S užkraunamas į registrą SS automatiškai iki programos vykdymo pradžios. Šis variantas yra tinkamesnis.

Klausimai

1. Kas tai yra ASSUME ir kokia jos paskirtis programos darbe?
2. Ar komanda ASSUME įtakoja programos ilgį?
   Taip
   Ne
3. Kas tai yra prefiksas ir koks yra jo vaidmuo programos darbe?

4. Kas yra atsakingas už segmentinių registrų formavimą?
   Sistema
   Programuotojas
   Programos vartotojas
5. Kokius žinote programinius segmentus?

PROGRAMOS STRUKTŪRA

Pilna programa assemblerio kalboje neturi griežtos struktūros, tačiau nedidelėm programom naudojami trys segmentai- komandų, duomenų ir steko. Tipinė schema

```
STACK SEGMENT STACK          ;steko segmentas
DB 128 DUP(?)
STACK ENDS
DATA SEGMENT                 ;duomenų segmentas
KINTAMŲJŲ IR KONSTANTŲ APRAŠYMAS
DATA ENDS
CODE SEGMENT                 ;komandų segmentas
ASSUME CS:CODE,DS:DATA,SS:STACK
START: MOV AX,DATA
        MOV DS,AX            ;DS registro užkrovimas
Kitos programos komandos
CODE ENDS
       END START             programos pabaiga, įėjimo taškas
```

Segmentų tarpusavio išdėstymas programoje gali būti bet koks, tačiau kad sutrumpinti komandose kreipinius pirmyn ir išvengti problemų dėl prefiksų rašymo, rekomenduojama komandų segmentą talpinti programos gale. Steko segmentas programoje aprašytas su parametru STACK , todėl pačioje programoje jau nebereikia užkrauti registro SS. Kaip žinia nereikia užkrauti ir registro CS. Todėl lieka tik vienas registras kurį reikia užkrauti - tai registras DS.

Nežiūrint į tai, kad programoje mes nevartojame Steko segmento, jį vis tiek reikia aprašyti, nes jį naudoja operacinė sistema, kai yra atliekami pertraukimai. Programos pabaigoje reikia nurodyti END direktyvą. Ji naudojama kad parodyti jog tai jau visi programos sakiniai ir visi kiti pasitaikantys už jos sakiniai nebepriklauso programai. Joje nurodomas taip vadinamas įėjimo taškas, t.y. žymė komandos nuo kurios turi pradėti dirbti programa. Visi sakiniai programoje turi priklausyti kuriam nors segmentui. Tarp segmentų galima nurodyti tik kokią nors informaciją.
Iki šiol kalbėta apie darbą su vienu komandų segmentu, kas būdinga mažoms programoms. Bendru atveju gali būti tiek komandų, kad netelpame viename segmente. Tuo atveju programoje aprašome keletą komandų segmentų.
Pvz.

```
C1 SEGMENT
ASSUME CS:C1….
START: MOV AX,0
…
JMP FAR PTR L
…
C1 ENDS
C2 SEGMENT
ASSUME CS:C2
L: INC BX
…
C2 ENDS
```

Kiekvieno komandų segmento pradžioje turi būti nurodyta direktyva ASSUME kur segmentiniam registrui CS nustatomas reikiamas segmento adresas. PK komandos, kuri turi būti vykdoma, adresas nustatomas per CS ir IP adresinę porą . CS nurodo segmento pradžią, o registre IP yra komandos poslinkis to segmento atžvilgiu. Bet kokio registro reikšmės pakeitimas keičia ir viso adreso pakeitimą. Jei keičiama tik IP reikšmė, tai reiškia kad perėjimas vyksta tame pat segmente. Tokie perėjimai vadinami artimais. Jei programoje yra keletas komandų segmentų tai reikia nurodyti perėjimus iš vieno segmento į kitą. Tokie perėjimai vadinami tolimais. Esant tokiems perėjimams keičiasi registrų CS ir IP reikšmės. CS nurodo segmento pradžią, o IP užrašomas poslinkis segmento viduje.

Stekas ir steko segmentas

Stekas – tai programos sritis laikinam bet kokių duomenų saugojimui.
Stekas - tai saugykla, darbas su kuriuo atliekamas pagal principą: paskutinis elementas, užrašytas į steką yra nuskaitomas iš steko pirmas. Tokiai saugyklai OA galima išskirti bet kokią vietą, bet jai yra taikomi du kriterijai:
Sritis neturi viršyti 64 kb.
Pradinis adresas turi būti kartotinis 16.
Kitaip sakant, ta sritis turi būti atminties segmentu. Ji ir vadinama steko segmentu. Jo pradžia užduodama segmentiniam registre SS. Stekas užpildomas iš viršaus žemyn. Pirmas elementas užrašomas pačiam dugbe, sekantis užrašomas ant jo ir t.t. Skaitant iš steko gaunasi, kad steko apačia (dugnas) yra fiksuota, o steko viršūnė visą laiką kinta. Kad žinoti tos viršūnės padėtį bet kuriuo metu, yra naudojamas dar vienas registras SP (nuoroda-taškas steke). Jame saugomas adresas tos lastelės, kurioje yra elementas, įrašytas į steką paskutinis. Dar tiksliau, SP yra tos lastelės poslinkis, t.y. jos adresas atskaičiuotas nuo steko segmento pradžios. Todėl absoliutinis steko viršūnės adresas užduodamas regisrtų poros SS:SP.
Pagrindinė steko ypatybė yra savotiškas duomenų išrinkimas, t.y. bet kuriuo metu steke yra prieinamas tik viršutinis elementas, toks kuris į steką pakliuvo paskutinis. Tik iškrovus iš steko viršutinįi elementą galime prieiti prie sekančio.
Steko elementai išdėstyti operatyvinėje atminty, skirtoje stekui, pradedant nuo steko dugno, t.y. nuo jo max adreso, ir užpildoma nuosekliai mažėjant steko viršūnės adresui. Viršutinis prieinamo elemento adresas saugomas steko viršūnės registre SP. Patalpinti bet kokią reikšmę į steką ar paimti bet kokią reikšmę iš steko galima su specialiomis komandomis. Padedant į steką yra sumažinama registro SP reikšmė per 2 ir patalpinamas operandas nurodytu adresu, kuris saugomas registre SP. Išėmus iš steko duomenis – jis neišsivalo, bet jame esantys elementai neprieinami, nes pasikeite SP reikšmė –ji padidėja per 2, o stekas laikomas tuščiu.
Steko elementai gali būti bet kokio dydžio: tai gali būti baitai, žodžiai, dvigubi žodžiai ir kt. Tačiau PK turimos įrašymo į steką ir nuskaitymo iš steko komandos dirba tik su žodžiais. Paprastai derinamasi prie komandų ir skaitoma, kad steko elementai yra vieno žodžio.
Reikia išskirti terminą “steko segmentas” ir “stekas (t.y. jo turinys)”
“steko segmentas”- reiškia atminties sritį, kurią potencialiai gali užimti steko duomenys, o “steaks”- reiškia duomenis, kurie duotu metu yra steke, t.y. baitai, pradedant nuo adreso SP iki steko segmento pabaigos. Visi duomenys, kurie yra virš adreso iš SP nesiskaito kaip steko duomenys. Kad rezervuoti stekui vietą OA, programoje reikia aprašyti atitinkamą programinį segmentą :

```
S SEGMENT STACK        Šis programos fragmentas išskiria
        db k dup(?)    k baitų OA stekui,
S ENDS                 t.y. steko segmentas
```

Vardų steko lastelėms nenurodome, nes priėjimas prie jų vis tiek vyks ne pagal vardus, o netiesiogiai per registrą SP. Dažniausiai neužduodamos ir tų lastelių pradinės reikšmės. Todėl aprašant steko segmentą, nurodome tik kiek baitų skiriame segmentui.
Prieš pradedant dirbti su steku, būtina užkrauti registrus SP ir SS: t.y. registras SS turi rodyti steko segmento pradžią, o SP turi rodyti steko viršūnę. Tai galima padaryti :jei aprašant steko segmentą su direktyva SEGMENT nurodysime parametrą STACK, tai

pradedant vykdyti programą abu šie registrai bus užkrauti automatiškai t.y SS registre bus patalpintas segmento pradžios adresas, o registre SP bus užrašytas steko dydis baitais - “k” (kiek jam išskirta aprašymo metu) - t. y. todėl, kad pradedant programai dirbti stekas turi būti tuščias, o “k” reikšmė ir nurodo tuščią steką, t.y. SP nurodo tuščią lastelę už steko.
Net jei mes nenaudojame programiškai steko, vis tiek jam būtina išskirti mažiausia 256 baitus. Jei mes naudojam programiškai steką, tai jam išskiriam vietos tie kiek reikia ir plius dar tuo 256 baitus.

Steko komandos:

Įrašymas į steką (vieno žodžio) PUSH op
galimos op reikšmės yra registras r16,
OA m16
sr bet koks segmentinis registras
Negalima nurodyti tiesioginės reikšmės. Tai galima atlikti naudojant tarpinį registrą.

→ PUSH op SP

SP POP op

Komanda PUSH veikia sekančiai: pirmiausia registro SP reišmė sumažinama 2, t.y SP pastumiama į viršų ir dabar jau rodo laisvą steko sritį, ir tada jau į ją įrašo operando reikšmę:
SP:=(SP-2)
Ši komanda vėlevėlių reikšmių nekeičia
Dvigubam žodžiui užrašyti reikės atlikti komandą du kartus, o užrašyti vieną baitą, reikia prieš tai ją išplėsti iki žodžio nesvarbu kokiu būdu.

Skaitymas iš steko (vieno žodžio) POP op
galimos op reikšmės yra:
registras r16,
OA m16
sr bet koks segmentinis registras (išskyrus CS)
Ši komanda išstumia iš steko viršūnės žodį ir patalpina jį į nurodytą operande vietą.
Ši komanda veikia sekančiai:
Iš lastelės, kurios adresas išskaičiuojamas iš registrų SS:SP poros, reikšmės ersiunčiama į operandą, ir tuo pat metu registro SP reikšmė padidinama per 2, t.y. viršūnė steko nustumiama žemyn t.y.
SP:=(SP+2)

Stekas gali būti naudojamas :

Registrų išsaugojimui - kai reikia atsiminti registrų reikšmes, kai tie registrai laikinai turi būti panaudojami kitiems tikslams, tai galim jo reikšmę laikinai išsaugoti steke, o po registro atlaisvinimo, jo reikšmę atstatyti nuskaitant iš steko.
Pvz. Reikia išsaugoti registrą CX

PUSH CX

POP CX

Persiūsti duomenis per steką

Dažnai jis naudojamas, kai reikia persiūsti duomenis iš vienos lastelės į kitą (kaip prisimenat to daryti tiesiogiai negalima, paprastai tam naudojamas tarpinis registras)

PUSH Y

POP X tolygu X=Y

Priėjimas prie steko elementų
Komandos PUSH ir POP leidžia prieti prie duomenų steko viršūnėje, bet kartais reikalingas priėjimas ir prie kitų steko elementų, kurie yra žemiau. Tarkim steke įrašyta trijų žodžių reikšmės w1,w2,w3
Reikia paimti w3 reikšmę (pačią apatinę) ir ją įrašyti į registrą AX. Jo adresas yra viršūnės adresas +4 Tai mes galime padaryti steko viršūnės adresą patalpindami į registrą BP, o po to jau imti reikiamą elementą nrodant poslinkį 4 t.y pagal adresą (BP+4)

```
MOV BP,SP
MOV AX,[BP+4]
```

Registrą BP naudojame kaip registrą modifikatorių, nes SP negali būti kaip modifikatorius. Tuo atveju jei nenurodytas joks segmentinis registras, pagal nutylėjimą jis priimamas kaip SS, o tai reiškia kad yra skaitomi duomenys iš steko segmento. Tačiau jei vietoj BP naudotume kitą registrą, pvz. BX, tai pagal nutylėjimą būtų imamas kaip segmento registras, registras DS, o tai reiškia, kad duomenys būtų skaitomi iš duomenų segmento.

Duomenų segmentas
Kaip ir bet kuris kitas programini segmentas jis turi savo vardą ir apiforminamas specialiasi operatoriais. Duomenų segmente yra aprašomi kintamieji ir kiti duomenys,kurie bus naudojami programos viduje. Duomenų aprašymui duomenų segmente yra naudojami operandai, kurie leidžia formuoti operatyvinėje atmintyje skirtingo tipo r skirtingo ilgio duomenis. Už jų panaudojimą programoje atsakingas programuotojas.

Komandų segmentas
Tai yra vieta, kur realizuojamas programos algoritmas, naudojant reikiamas komandas. Šio segmentoviduje taip pat aprašomos paprogramės, jei jos naudojamos programoje.

Paprogramės –Procedūros
Dažnai programose, ypač dideliuose, reikia keletą kartų spręsti tą pčią užduotį ir todėl reikia daug kartų aprašyti tokią pačią komandų seką, sprendžiančią šią užduotį. Kad išvengti pasikartojančių komandų grupių, tokias komandų grupes aprašo vieną kartą ir apiformina atatinkamu būdu, o po to reikiamose vietose tiesiog perduoda valdymą į tas komandas, ir kai jos atidirba, grąžina valdymą atgal. Tokia komandų grupė, kuri sprendžia tam tikrą užduotį ir kuri organizuota tokiu būdu, kad ją galima būtų naudoti neribotą skaičių kartu ir į ją būtų galima kreiptis iš bet kurios programos vietos, vadinama paprograme. Jos atžvilgiu kita programos dalis vadinama pagrindine programa. Lengva suprasti, kad paprogrames tai procedūrų analogas aukšto lygio programavimo kalbose. Pati paprogramė – tai analogas procedūros aprašymo, o kreipinys į paprogramę yra analogas procedūros operatoriaus. Realizacija paprogrames assembleryje yra sudėtingesnė nei procedūros aprašymas. Esmė čia, kad aukšto lygio kalbose už mus tai daro transliatoriai, o programuojant assembleryje mums niekas nepadeda, todel čia reikia atlikti viską pačiam programuotojui.

kur talpinti paprogrames?
Bendru atveju kalbant jas galima talpinti bet kur. Bet reikia prisiminti, kad pati savaime paprogramė neturi būti vykdoma, o turi būti vykdoma tik tada, kai yra į ją kreipinys. Todel talpinti ją reikia taip, kad į ją atsitiktinai nebūtų galima pakliūti. Jei yra keletas paprogramių , tai jas galima sudėti greta. Paprastai paprogramės dedamos arba gale komandų segmento už paskutines komandos, arba komandų segmento pradžioje prieš komandą nuo kurios programa turi pradėti veikti. Didelėse programose kartais paprogramės iškeliamos į atskirą segmentą.

kaip apiforminti paprogrames?
Bendru atveju komandų grupę, kuri bus kaip paprogramė, galima niekaip neišskirti. Tačiau asembleryje priimta paprogrames apiforminti ypatingu būdu- kaip procedūrą. Tai yra tam tikru būdu naudinga. Paprogramės kaip procedūros apiforminimas atrodo taip:
<proceduros vardas> PROC <parametrai>
Procedūros kūnas
<procedūros vardas> ENDP
Kaip matome, procedūros kūnas eina po direktyvos PROC, o po jos eina direktyva ENDP. Tose abiejose direktyvose nurodomas tas pats vardas- vardas, kurį mes davėme paprogramei. Reikia atkreipti dėmesį, kad direktyvoje PROC po vardo nededame dvitaškio, bet šis vardas skaitomas žyme, kurią turi pirmoji procedūros komanda. Šį vardą galima naudoti perdavimo komandoje ir perdavimas bus vykdomas į šią komandą. Direktyvoje PROC yra parametras – tai gali būti **NEAR** (artimas) arba **FAR** (tolimas). Parametro gali ir nebūti, tada priimama kad jis yra **NEAR**. Esant parametrui **NEAR** arba nesant jokiam parametrui, procedūra yra vadinama “artima”, o esant **FAR** parametrui- “tolima”. Į “artimą” procedūrą galima kreiptis tik iš to paties komandų segmento kuriame ji aprašyta, ir negalima kreiptis iš kitų komandinių segmentų, o į tolimą procedūrą galima kreiptis iš bet kokių komandinių segmentų. Assembleryje vardai ir žymes, aprašyti proceduroje, nėra lokalizuojami jos viduje, todėl jie turi būti unikalūs ir negali sutapti su kitais vardais, naudojamais programoje.

procedurų iškvietimas ir grįžimas iš jų
Kaip iškviesti procedūras ir grižti iš jų? Programuojant aukšto lygio kalbomis, procedūrų vykdymui pakanka tik parašyti procedūros vardą ir faktinius parametrus. Tada mūsų nejaudina kaip procedūra pradės dirbti ir kaip ji griš į pagrindinę programą. Assembleryje visus valdymo perdavimus iš programos į procedūrą ir atvirkščiai reikia realizuoti patiems. Čia yra dvi problemos: kaip iš pagrindines programos perduoti valdymą procedūrai ir kaip iš jos grižti į pagrindinę programą. Pirmoji problema sprendžiama paprastai. Pakanka tik perduoti valdymą į pirmą procedūros komandą, t.y. nurodyti perdavimo komandoje proceduros vardą. Sunkiau yra realizuoti grižimą. Esme ta, kad į procedurą galima kreiptis iš ivairių programos vietų, vadinasi grižti reikia į skirtingas vietas. Pati procedūra nežino kur ji turi perduoti valdymą, kai baigs savo darbą.

Parametrų perdavimas parametrų perdavimas per registrus
Aukšto lygio kalbose kad perduoti parametrus paprogramei užtenka juos tik užrašyti į operatorių, kuris kviečia procedurą. Asembleryje parametrai ir procedūros rezultato perdavimas yra kitas. Asembleryje nera fomalaus dalinimo į procedūras ir funkcijas, tačiau pagal turinį yra dalinama į procedūras ir funkcijas. Perduodami faktiški parametrai skirtingai. Paprasčiausias būdas yra registrai. Pagrindineje programoje faktiniai parametrai yra užrašomi į kokius nors registrus, o procedūra toliau jais naudojasi. Analogiškai galima pasielgti ir su rezultatais, jei tokie yra. Procedūra užrašo rezultatus, o programa vel naudojasi. Kokie bus tam naudojami registrai priklauso nuo paties programuotojo.
parametrų perdavimas per reikšmes
Pvz. Reikia išskaičiuoti c=max(a,b)+max(5,a-1), kur visi skaičiai – skaičiai su ženklu ir yra žodžio tipo. Išskaičiavimą max(x,y) aprašysim kaip procedurą funkciją, tuo pačiu susitarsime apie sekantį:
pirmas parametras (x) programai perduodamas per AX registrą, o kitą (y) –per registrą BX, o rezultatą (max) procedūra turi grąžinti per registrą AX. Esant tokioms sąlygoms procedura ir pagrindines programos fragmentas atrodys taip:

```
; procedura : AX=max(AX,BX) ;PAGRINDINe PROGRAMA
MAX PROC FAR MOV AX,a ;AX:=a
CMP AX,BX MOV BX,b ;bx:=b
JGE MAX1 CALL MAX ;AX:=MAX(a,b)
MOV AX,BX MOV c,AX ;Išsaugome AX
MAX1: RET MOV AX,5 ;AX:=5
MAX ENDP MOV BX,a
DEC BX ;BX:=a-1
CALL MAX ;AX:=MAX(5,a-1)
ADD c,AX ;c:=max(a,b)+max(5,a-1)
```

parametrų perdavimas persiuntimo būdu
Tegu programoje yra kreipinys, kai kintamieji A ir B (kurie yra teigiami skaičiai) ir reikia priskirti jiems kokias tai reikšmes, tai yra persiūsti duomenis. Kad kažką įrašyti i lastelę reikia žinoti tos lasteles adresą. Todėl jei priskyrimas numatytas atlikti procedūroje, procedūrai reikia nurodyti tų lastelių adresą ir tai turi atlikti pagrindinė programa. Tokiu būdu parametro perdavimas persiuntimo būdu reiškia adreso perdavimą, kuris atitinka faktinį parametrą. Kaip perduoti tą adresą Per registrą: pagrindinė programa užrašo į kokį nors registrą adresą faktinio parametro, o procedūra jį iš ten pasiima. Koks tai registras? Bendru atveju bet koks, bet geriau jei tai bus registras modifikatorius, t.y. BX, BP, SI, DI kadangi procedūrai reikės atlikti adresų modifikaciją.
Tegu mūsų procedūrai D mes pasirinkome registrą BX. Tai reiškia kad iki jos darbo pradžios registre BX bus tos lasteles adresas (A arba B) , kurios turinį procedūra turi pakeisti. Tokioje situacijoje pasiekti šią lastelę galima naudojant konstrukciją [BX].

```
; Pagrindine programa ;Procedura
... D PROC
LEA BX,a ;;BX:=Adresas a PUSH CX ;išsaugom CX
CALL D ; procedura D MOV CL,4
```

```
LEA BX,b ;;BX:=Adresas b SHR WORD PTR [BX], CL
CALL D POP CX ;atstatom registr¹ CX
... RET
D ENDP
```

registrų išsaugojimas procedūroje

Kaip matome, procedūroje D bus naudojamas registras CL, kuriame procedūra patalpina poslinkio dydį. Ar gali procedūra keisti registro turinį? PK ne taip jau daug registrų, o jie yra naudojami daugelyje komandų. Didelė tikimybė, kad procedūroje ir programoje gali būti reikalinga naudoti tuos pačius registrus. Todėl reikia , kad procedūra prieš naudodama registrus įsimintų jų turinį. Tai galima atlikti kai procedūros pradžioje jų turinys saugojimui užrašomas steke, ir po to procedūra registrus gali naudoti savo nuožiūra. Prieš grįžtant į pagrindinę programą tuos registrus reika atstatyti perrašant į juos reikšmes iš steko. Yra viena ypatybė, netgi jei mums reikia išsaugoti tik vieną registro baitą mes turime įsiminti viso registro turinį, nes stekas gali skaityti ir rašyti duomenis tik žodžiais. Yra specialios komandos PUSHA ir POPA, kurios įsimena ir atstato visus bendro naudojimo registrus.

sudetingo tipo parametrų perdavimas

Tai gali būti masyvai ar struktūros. Tuo atveju geriau perduoti ne pačius duomenis, o adresą kur tie duomenys yra. Tada bus galima prieiti iš procedūros prie bet kurių nurodytų duomenų. Tai atliekama netgi tada kai nereikia tų duomenų keisti.

Pvz.

```
X DB 100 Dup(?)
Y DB 25 Dup(?)
```

Reikia į DL užrašyti sumą maksimalių reikšmių iš masyvų X ir Y atatinkamai, t.y. DL=max(X[I])+max(Y[I]) Šiuo atveju du kartus reikia išrinkti maksimalią reikšmę iš masyvų X ir Y atatinkamai. Tam sudarysime procedūrą MAX. Pradinis masyvo adresas perduodamas per registrą BX, o elementų kiekis masyve per registrą CX, o rezultatas grąžinamas per registrą AL.

```
;procedura MAX
MAX PROC
PUSH BX
PUSH CX
MOV AL,0
MAX1: CMP [BX],AL
JLE MAX2
MOV AL,[BX]
MAX2: INC BX
LOOP MAX1
POP CX
POP BX
RET
MAX ENDP
; Pagrindinė programa
LEA BX,X ;masyvo X adresas
MOV CX,100 ;CX yra masyvo elementų skaičius
CALL MAX ;AX=max(X[I])
```

```
MOV DL,AL ;isimenam AL
LEA BX,Y ;masyvo Y adresas
MOV CX,25 ;elementu kiekis masyve
CALL MAX ;AX=max(Y[I])
ADD DL,AL
```

parametrų perdavimas per steką

Parametrų peradvimas per registrus yra patogus metodas ir dažnai naudojamas. Tačiau tai yra patogu, jei nėra daug parametrų. Jei parametrų daug, tai registrų gali ir neužtekti. Tuo atveju naudojame kitą parametrų perdavimo būdą t.y. jie gali būti perduoti per steką. Pagrindinė programa patalpina parametrus į steką, o procedūra juos iš ten nuskaito. Tai atlikti galima įvairiai. Taigi procedūra P naudoja k parametrų. Sutarsime, kad prieš kreipiantis į procedūrą parametrai užrašomi į steką. Kokia tvarka? Tai nustato programuotojas. Mes manysime, kad parametrai į steką užrašomi iš kairės į dešinę. Pradžioje užrašomas 1-as elementas, po to kitas ir t.t. Jei kiekvienas parametras yra žodis, tai pagrindinės programos komandos, realizuojančios kreipimąsi į procedurą , bus sekančios:

```
; kreipinys į procedurą
push a1
push a2
...
push ak
call p
```

Kai pradeda dirbti procedūra iš kart kyla problema kaip jai pasiimti parametrus. Tai yra problema, kai reikia pasiekti steko elementus jų neišimant iš steko. Tai galima realizuoti naudojant registrą BP, t.y. nusiūsti į šį registrą steko viršunės adresą (registro SP reikšmė), o vėliau naudoti išraišką [BP+I]. Šiuo atveju mes gadiname registro BP reikšmę. Todėl pradžioje jį reikia išsaugoti ir tik po to į jį persiūsti SP reikšmę. Taigi procedūra turi prasidėti nuo pagrindinių komandų.

```
; pradiniai proceduros veiksmai
P Proc
Push BP
MOV BP,SP
...
tolimesnės procedūros komandos
```

Reikia atsiminti tvarką, kokia parametrai buvo padeti į steką. Procedūros pabaigoje turi eiti galinės komandos. Tai komandos, kurios atstato registro BP reikšmę ir dar reikia išvalyti steką nuo parametrų. Steko valymą gali atlikti ir pagrindinė programa. Tam po komandos CALL P reikia ivykdyti komandą ADD SP,2*k. Bet geriau jei tai atliekama procedūroje dėl to, kad į procedūrą galima kreiptis iš ivairių programos vietų ir del to komandą ADD reiktų atlikti keliose pagrindinės programos vietose. Taigi procedūra turi išvalyti steką nuo parametrų ir tik tada grąžinti valdymą pagrindinei programai. Tam yra numatytas komandos RET variantas, kur operando pagalba galime nurodyti kiek baitų steke reikia išvalyti. Operando reikšmė traktuojama kaip skaičius be ženklo. Ši komanda pirmiausia iš steko pasiima grįžimo į pagrindinę programą adresą, tada steke išvaloma tiek baitų kiek yra nurodoma ir tik po to valdymas perduodamas į pagrindinę programą. Iš esmės paprasta RET yra RET 0. (valymo sritis nurodoma baitais o ne žodžiais).

KOMANDŲ SISTEMA IR JŲ KLASIFIKAVIMAS
Persiuntimo komandos
Aritmetinės komandos
Lyginimo komandos
Ciklo organizavimo komandos

Duomenų persiuntimo komandos

Bendros paskirties
Darbui su steku
Perkodavimo

MOV op1,op2
Komanda persiunčia baito ar žodžio ilgio duomenis iš op2 į op1.(registrą arba OA). Negalima persiūsti duomenų iš OA į OA tiesiogiai. Taip pat negalima persiūsti duomenų iš vieno segmentinio regitro į kitą segmentinį registrą. Taip pat nealimakeisti registro CS reikšmės. To priežastis yra ta, kad CS ir IP reikšmės apsprendžia adresą komandos, kuri turi būti vykdoma sekanti, tai reiškia kad pakeitus kurio nors registro reikšmę mes jau atliekam valdymo perdavimą.
Kartais iš operandų neina nustayti kiek reikia persisti baitų.
Pvz. MOV [SI],0 neaišku ar 0 yra baitas aržodis. Todėl neaišku, kiek baitų išvalys nuo adreso, užrašyto register SI. Todėl tokiu atveju reikia patikslinimo komadoje. Tam galime panaudti operatorių PTR

<Tipas>	PTR	<išraiška>	
<Tipas> gali būti		BYTE	nustato baitą
		WORD	nustato žodį
		DWORD	nustato dvigubą žodį
<išraiška>		konstanta	PTR nustato kas tai yra –baitasar žodis
		Adresinė konstanta	

BYTE	PTR	0	0 traktuojamas kaip baitas
WORD	PTR	0	0 traktuojamas kip žodis
DWORD	PTR	0	0 traktuojamas kaip dvigubas žodis

```
MOV     BYTE PTR[SI],0
MOV     [SI],BYTE PTR 0

MOV     WORD PTR[SI],0
MOV     [SI],WORD PTR 0
```

Su PTR galime ne tik patikslinti bet ir pakeisti duomenų tipą

PVZ.
Z DW 1234
3412
Z Z+1
Reikia tik dalį žodžio išvalyti

MOV BYTE PTR Z,0
Bus išvalomas tik pirmas baitas

Prie persiuntimo komandų galima priskirti ir adreso persiuntimą.

LEA r16,A Į nurodytą registrą įrašomas adresas nurodytos žymės

X DW 88
Y DW X
Reikia X adresą patalpinti registre.

Jei MOV BX,X, tai BX=88
MOV BX,Y Tai bus X adresas. Bet galima ir kitaip

LEA BX,X Dažnai naudojama formuojant išvedimo adresą. LEA komanda pati paskaičiuoja adresą. Vėlevėlių nekeičia. Antras operandas gal būti ir baitas ir žodis, nes imama ne jo reikšmė, o adresas,bet negali būti konstanta arbakitas registras.
Vietoj LEA galima naudoti komandą
MOV SI,OFFSET Kintamasis
LEA SI,Kintamasis

Aritmetinės komandos

Skaičių be ženklo sudėtis ir atimtis
Skaičiavimai atliekami dvejetainėje sistemoje. Je rezultatas didesnis nei jam numatytas operando dydis gali sutalpinti, kai kuriuose kompiuteriuose fiksuojama klaida ir programos darbas nutraukiamas.

Personaliniuose kompiuteriuose klaida nefiksuojama, o pernešimo vienetas atmetamas ir rezultate gauname iškreiptus duomenis, tačiau tuo patmetu fiksuojama pernešimo vėliavėlės reikšmė CF=1.
Analizuojant vėliavėlės reikšmę galima keisti programos darbą. Toks sumavimas vadinamas sumavimu pagal modulį 2^k, kai atmetamas perpildymo vienetas.

Skaičių be ženklo atveju
Suma(x,y)=x+y

Mod 2^k =x+y jei X+Y<2^k CF=0
=X+Y+2^k jei X+Y>=2^k CF=1
Tokia pat problema yra ir skaičių be ženklo atimties atveju. Ką daryti X-Y jei X<Y
Tuo atveju gunamas neigiamas skaičius, o tai jau skaičius su ženklu.

X>=Y vyksta paprasta atimtis
X<Y X pasiskolinamas vienetas, t .y. prie X pridedamas 2^k, o po to atliekama atimtis.
Gautas rezultatas paskelbiamas rezultatu papildomam kode (256+1)
1-2=(2^8+1)-2=255 (papildomam kode) = (-1) CF=1 klaida nefisuojama. Tai yar atimtis pagal modulį 2^k.

Skirtumas (x,y)=(x-y) mod 2^k=x-y x>y CF=0
=(2^k+x)-y x<y CF=1

Sudėtis iratimtis skaičių be ženklo yra atliekamas pagal modulį 2^k. K nusako OA lastelės dydį .Klaida fiksuojamaCF vėliavėlėje.

Sudėtis ir atimtis skaičių su ženklu

Vykdoma papildomam kode.Tada jie sudedamiir atimami kaip skaičiai be ženklo. Po to rezultatas nagrinėjamas kaip papldomas kodas skaičiaus su ženklu.

3+(-1) = (3)+(256-1)=3=255(mod 256)=258(mod 256)=2
Komandos:

ADC op1,op2 sudedami du skaičiai ir CF vėliavėlės reikšmė.
Op1=op1+op2+CF

SBB op1,op2 atimami du skaičiai
Op1=op1 -op2-CF reikšmė pasiskolinimui

Šių komandų pagalbagalima sudėti arba atimti įvairių dydžių skaičius.

ADD op1,op2 op1:=op1+op2 skaičių sudėtis

SUB op1,op2 op1:=op1-op2 skaičių atimtis

OP1	OP2
R8	i8,r8,m8
M8	i8,r8
R16	i16,r16,m16
M16	r16

Bendrai komandos veiksmus galima aprašyti

Op1:=op1*op2 operandų tipas turi būtivienodas
Skaičiai gali ūti su ženklu arba be ženlo.
Keičiasi vėliavėlės

CF pernešimo
OF perpildymo
SF ženklo
ZF nulio

Analogiškai veikia komandos
INC op prideda vienetą
DEC op atima vienetą
Jos taip pat dirba su baitu arba su žodžiu., be to **OP** gali būti registras arba OA lastelė.

NEG op supranta operandą kaip skaičių su ženklu ir keičia jo ženklą į priešingą.

Daugybos komandos

Skaičių be ženklo daugyba

MUL op
Skaičių su ženklu daugyba

IMUL op

Vieno iš operandų vieta yra fiksuota,t.y
AL skaičiams viename baite
AX skaičiams žodyje

Baito dydžio skaičių daugyba
AX:=AL*op op:r8,m8

Žodžio dydžio skaičių daugyba
(DX,AX)=AX*op op:r16,m16

Op gali būti registre ar operatyvinėje atmintyje, bet negali būti kaip duomenys komandoje. Rezultatui skiriama dvigubai daugiau vietos, nei užima dauginamasis. N ženklų daugybos rezultatas yra 2n..
Sudauginus baitus, rezultatas talpinamas žodyje.
Sudauginus žodžius, rezultatas talpinamas dvigubame žodyje.

PVZ.
N*2 MOV AL,2
MUL N

MOV AX,2

```
MOV  BX,-1
IMUL  BX
```

Kartais rezultatui formuoti užtenka tokio paties dydžio lastelių kaip ir dauginamieji.Kartais tai žinoma iš anksto, bet gali būti ir nežinoma.Tuo atvejunustatyti ar rezultatui reikiato paties ilgio ar dvigubo ilgio, galima nustatyti nalizuojant vėliavėles.

CF=OF=1 jei reikia dvigubo formato
CF=OF=0 jei uženka to paties formato
CF=0 jei rezultatas yra tik register AL, arba AX dėl baito ar dėl žodžio atatinkama.
CF=1 tai rezultatui reikia dvigubo formato,t.y. AX dėl baito, (DX,AX) dėl žodžio.

Dalybos komandos

Skaičių be ženklo dalyba
DIV op

Skaičių su ženklu dalyba
IDIV op

Dalinant skaičių iš baito ar žodžio daliklis gali būti registre arba operatyvinėje atmintyje. Dalijamojo vieta fiksuota ir priklauso nuo operando dydžio.Rezultatas yra sudarytas iš dviejų dalių: sveikasis skaičius ir liekana.Jų vieta priklauso nuodalijamojo ir daliklio, betyra fiksuota ir jo nurodyti nereikia. Ėlevėlės nesikeičia, bet yra ypatingas atvejis, jei daliklis yra 0, t.y. dalyba iš nulio.
Dalijant žodį iš baito
Dalijamasis turi būti AX, o daliklis užimti vieną baitą
Dalijant dvigubą žodį iš žodžio , dalijamasis turibūtiužrašytas per du registrus DX ir AX.
DX vyresnieji skaičiai, AX jaunesnieji skaičiai.

Dalijamais	Daliklis	Sveikoji Dalis	Liekana
Žodis AX	Baitas registre Arba OA	Baitas registre AL	Baitas registre AH
DIV op	AH AL sveikoji dalis Liekana		
Dvigubas Žodis (DX,AX)	Žodis registre Arba OA	Žodis registre AX	Žodis registre AL
DIV op	DX AX sveikoji dalis Liekana		

Lyginimo komanda ir sąlyginis valdymo perdavimas
Jei valdymas perduodamas komandai tik esant tam tikrai patenkintai sąlygai, o priešingu atveju ne, tai vadinama sąlyginiu valdymo perdavimu. Valdymo perdavimas vykdomas per d etapus:
Lyginamos reikšmės ir kaip rezultatas formuojamos vėlevėlių reikšmės
Priklausomainuo vėlevėlių reikšmių valdymas perduodamas arba ne.
Lyginimui naudojama speciali komanda.

Lyginimo komanda

CMP op1,op2
Ši komanda ekvivalenti komandai SUB, t.y. iš op1 atimamas op2. Skirtumas tiktoks, kad nei op1 nei op2 nesike2ia,o tik formuojama vėlevėlių reikšmė ir taip formuojamas rezultatas.
C=0 jei op1>op2 skaičiai be ženklo
C=1 jei op1<op2

O=0 jei op1>op2 skaičiai su ženklu
O=1 jei op1<op2

Z=1 jei op1=op2

Valdymo perdavimo komandos

Valdymo perdavimo komandų yra labia daug. Bendra forma yra
Jxx žymė komandos adresa kuriai perduodamas darbas
Komanda **Jxx** nustato vldym perdavimo sąlygas
Perdavimo komandos skirstomos į tris grupes

- Komandos rašomos iš karto po lyginimo komandos. Komandos kodas nustato sąlygą, kuriai esant patenkintai, vykdomas valdymo perdavimas. xx kodai komandoje yra sekantys:

E equal(lygu)
N not(nelygu,neigimas)
G grater(daugiau)
L less(mažiau)
A above(daugiau, aukščiau)
B below(žemiau,mažiau)
Ta pati sąlyga gali būtisuformuota skirtingais būdais. Pvz. **Mažiau** atatinka **Ne daugiau** ir panašūs dariniai. Sčlygos formavio būdą renkasi programuotojas.

JE op1=op2 ZF=1 bet kokiems skaičiams
JNE op1<>op2 ZF=0

Skaičiams su ženklu

JL/JNGE	op1<op2	SF<>OF	
JLE/JNG	op1<=op2	SF<>OF	Z=1
JG/JNLE	op1>op2	SF=OF	Z=0
JGE/JNL	p1>=op2	SF=OF	

Skaičiai be ženklo

JB/JNAE	op1<op2	C=1	
JBE/JNA	op1<=op2	C=1	Z=1
JA/JNBE	op1>op2	C=0	Z=0
JAE/JNB	p1>=op2	C=0	

- Komandos rašomos po kitų komandų, kurios reaguoja į vėlevėlių reikšmes, bet ne po lyginimo komandų. Nekeičiamos vėlevėlių reikšmės. Komandoje nurodoma vėlevėlės raidė, jei valdymas turi būti perduotas, kai vėlevėlė įjungta ir su N kai vėliavėlė turi būti išjungta:

JZ	Z=1	JNZ	Z=0
JS	S=1	JNS	S=0
JC	C=1	JNC	C=0
JO	O=1	JNO	O=0
JP	P=1	JNP	P=0

Šiai grupei priklauso kmandos, kurios tikrina registro CX reikšmę

JCXZ žymė

Valdymas perduodamas, kai CX=0
Visos valdymo perdavimo komandos atlieka trumpą perdavimą. Tai maždaug 30-40 komandų diapazonas. Jei reiki ilgo perdavimo, tai kombinuojam su besąlyginiu valdymoperdavimu:

```
CMP   AX,BX
JNE   L
JMP   M
L:
```

Ciklų organizavimo komandos

Tai tam tikrą kartą atliekamų komandų seka. Išėjimas iš ciklo gali būti sąlyginis ar besąlyginis., t.y. gali būti nustatytasgriežtas ciklų skaičius arba jis gali pasibaigti susidarius tam tikroms sąlygoms.

LOOP	sąlyginiam ir
LOOPE	besąlyginiam
LOOPNE	ciklui organizuoti

LOOP **Žymė** kur bus grąžinamas valdymas, t.y. ciklo pradžios adresas.

Instrukcija pagal nutylėjimą mažina registro CX eikšmę vienetu ir tuoj pat lygina su nuliu.
Jei CX<>0 tai valdymas perduodamas nurodytai žymei, jei CX=0 tai išeinama iš ciklo.
Pvz.

```
MOV  CX,10                MOV  CX,10
L:                        L:

LOOP L                    DEC  CX          Tos trys komandos
                          CMP  CX,0        atitinka komandą LOOP
JNE  L                    bet LOOP dirba grečiau
```

LOOP komandos trūkumai

- Atlieka tiktrumpą perėjimą
- Ciklui organizuoti reikalingas konkretus registras CX
- Bent kartą ciklas turi prasisukti, nes LOOP visada rašoma gale ciklo

Galima ciklą užbaigti ir anksčiau

LOOPE kai dar CX nelygus nuliui, bet cikle pasitaikė komanda, kuri vykdoma nustatė ZF reikšmę 1.

Yra numatytas konkretus ciklų skaičius, bet taippat galimas išankstinis išėjimas iš ciklo. Išėjimas iš ciklo dviem atvejais:kai CX=0 arba ZF=0. Tai galima nustatyti po ciklo.

ADRESAVIMO BŪDAI

Komandose operandai gali būti pateikiami įvairiai:
Operandai gali būti pateikti pačioje komandoje
Jie yra komandos dalis. Tiesioginiai duomenys gali būti tik antras operandas. Pirmu operandu gali būti lastelė OAarba registras. Tiesioginis perandas gali būti baito,žodžio arba dvigubo žodžio. Jis gali būti užrašomas per konstantą ar išraišką.
Operandai yra patalpini registruose. Komandoje nurododmas registro vardas. Registrų vardai yra bendros paskirties registrų vardai arba segmentiniai regisrai.
Operandai yra OA lastelėje. Sunkiausiai aprašomas bet lanksčiausias būdas. Yra du adresavimo būdai:

Tiesioginis
Šalutinis

Šalutinis adresavimas

Bazinis adresavimas (kitaip sakant registrinis bazinis adresavimas)
Bazinis adresavimas su poslinkiu
Indeksinis adresavimas su poslinkiu
Bazinis indeksinis adresavimas
Bazinis indeksinis adresavimas su poslinkiu

Operandu taip pat gali būti :
Įvedimo-išvedimo portas dėl išorinių įrenginių. Adresinė erdvė šiuo atveju iki 64kb. Kiekvinam įrenginiui šioje erdvėje skirtas adresas. Konkreti šio adreso reikšmė vadinasi portu. Fiziškzi įvedimo-išvedimo portui atitinka aparatinis registras. Priėjimas prie šių registrų yra vykdomas per komandas IN ir OUT.
Operandas gali būti steke.
Komandos gali turėti vieną ar du operandus. Daugumaikomandų reikia dviejų operandų. Vienas iš jų yra šaltinis,o antras priėmėjas. Svarbu kad vienas operandas gali būti registre arba OA, o kitas būtinai registre arba pačioje komandoje. Tiesioginis operandas gali būti tik šaltinis. Dviejų operandų komandose gali būti sekantys deriniai:

Registras-registras
Registras-OA
Tiesioginiai duomenys- registras
Tiesioginiai duomenys-OA

Išimtų sudaro komandos

Darbui su stku
Daugybos ir dalyos komandoms
Darbui sueilutėmis

Tiesioginis adresavimas

Tai yra pats pprasčiausias OA adresavimo būdas. Adresas imamas tiesiai iš laukokomandoje. Šis laukas yra baito, žodžio ar dvigubo žodžio. Ši reikšmėaprašo baitą, žodį ar dvigubą žodį duomenų segmente.

Santykinis tiesioginis adresavimas
Naudojamas sąlyginio valdymo perdavimokomandose, nurodant sąlyginį perėjimo adresą.

```
Pvz.    JE      M1- perdavimo adresas
        MOV     AL,2

M1:     Komanda, kuriai perduodamas valdymas
```

Absoliutinis tiesioginis adresavimas

```
Data    Segment
Per1    DW      5

Data    Ends
Program         Segment


                MOV   AX,Per1         tiesioginisadresavimas

Program         Ends
```

Šalutinis bazinis adresavimas

VALDYMO PERDAVIMAS

Tolimi perejimai

Iki šiol dirbome su vienu programiniu segmentu. Tai yra būdinga trumpom programom. Bendru atveju programoje gali būti tiek daug komandų, kad jos netelpa viename segmente. Tuo atveju programoje aprašome kelis komandinius segmentus.

```
C1 Segment
Assume CS:C1
Start: Mov ax,0
Jmp Far Ptr L
C1 ENDS

C2 SEGMENT
ASSUME CS:C2
L: INC BX
C2 ENDS
```

Kiekvieno segmento pradžioje turi buti nurodyta komanda ASSUME, kurioje segmentiniam registrui CS priskiriamas konkretus šio segmento atitikmuo. Priešingu atveju sutikęs pirmąją žymę transliatorius fiksuos klaidą.

Priminsime, kad PK komandos (kuri turi būti vykdoma) adresas užduodamas registrų CS ir IP pora . CS nurodo atminties pradžią kur yra komanda, o registre IP yra poslinkis iki tos komandos, atskaitytas nuo segmento pradžios. Bet kurio iš tų regisrtų turinio pasikeitimas yra ne kas kitas kaip perejimas, nes keičiasi vykdomos komandos adresas.

Jei keičiasi tik IP, tai reiškia kad perdavimas yra tame pačiame segmente. Tokie perėjimai vadinami artimais arba vidiniais. Jei programa nedidele ir visos jos komandos telpa viename segmente, tai kitokie perejimai ir reikalingi. Bet jei programoje yra keletas komandų segmentų, tai atsiranda poreikis organizuoti perejimus iš vieno segmento į kitą. Tokie perejimai vadinami tolimaisiais arba tarpsegmentiniais. Esant tokiems perdavimams keičiasi registro CS reikšmė ir registro IP reikšme. CS nustatomas i segmento C2 pradžią, o i IP užrašomas poslinkis iki žymes šiame segmente. PK numatytos komandos, realizuojančios tokius tolimus perėjiimus, tačiau jos visos yra besąlyginio perdavimo komandos.

Sąlyginio perdavimo komandos visada yra artimos.

Asembleryje tokio perdavimo tolimo perdavimo komandose kaip mnemokodas visada naudojamas tas pats JMP, kaip ir esant artimam perdavimui, bet naudojami kiti operandų tipai. Tolimo perdavimo komandos velevelių nekeičia.

Tiesioginis tolimas perdavimas JMP FAR PTR <žyme>

Čia **FAR** nurodo asembleriui, kad žymė tolima, kad ji yra kitame komandiniame segmente. Pagal šią komandą registras CS nustatomas į segmento pradžią, kuriame yra žymė, o į registrą IP užrašomas poslinkis nuo segmento pradžios iki tos žymės.

CS:=seg<žyme>; IP=offset <žyme>

Taip ankstesniam pavyzdyje nurodytas tplimas perejimas į komandą, kurios žymė yra L.

Šalutinis tolimas perejimas JMP m32

Šioje komandoje nurodomas adresas dvigubo žodžio, kuriame turi būti absoliutinis perejimo adresas užduotas kaip pora seg:ofs, ir užrašytas "perverstam" pavidale: t.y. ofs turi buti užrašytas adresu m32, o segmento numeris adresu m32+2. Komanda perduoda valdymą pagal šį absoliutinį adresą, t.y. užrašo į registrą CS dydį seg, o į registrą IP ofs reikšmę:

CS:=[m32+2]; IP:=[m32]

Pvz.

X dd L; X: offset L, X+2 :seg L

JMP X; per4jimas 5 L (CS:=seg L, IP:=offset L)

Užrašant šalutinio perdavimo komandas reikia elgtis atsargiai. Jeigu komandoje nurodytas vardas X , kuris aprašytas iki šios komandos, tai sutikęs ją, assembleris jau žino kad X reiškia dvigubą žodį ir vadinasi teisingai supras kad šioje komandoje turima omenyje tolimas šalutinis perdavimas. Bet jei X bus aprašytas vėliau, tada assembleris, sutikęs perdavimo komandą, nežinos šiuo atveju kokio pobūdžio turimas omenyje perdavimas- ar tai artimas perdavimas pagal žymę ar tai artimas šalutinis perdavimas, ar tai tolimas šalutinis perdavimas. Tokioje situacijoje assembleris įitaria, kad nurodytas vardas – tai esamo komandų segmento žymė ir formuoja artimo perdavimo mašininę komandą. Vėliau, kai bus nustatyta, kad šiame segmente nėra tokios žymės, assembleris fiksuos klaidą. Kad to išvengti, reikia valdymo perdavimo į priekį atveju aiškiai nurodyti, kad vardas nurodo dvigubo žodžio kntamąji. Tam naudojame operatorių PTR.

JMP DWORD PTR X

Kaip buvo pastebėta, tokio pobūdžio patikslinimus reikia naudoti ir esant artimam perdavimui. Jei viską apibudinsime viską kas susiję su besąlyginio perdavimo į priekį komandomis, tai gausime tokią taisyklę: jei X yra nuoroda į priekį, tai besąlyginio perdavimo komandą reikia užrašyti sekančiai:

JMP X artimas tiesioginis ilgas perdavimas

JMP SHORT X artimas tiesioginis trumpas perdavimas

JMP FAR PTR X tiesioginis tolimas perdavimas

JMP WORD PTR X artimas šalutinis perdavimas

JMP DWORD PTR X tolimas šalutinis perdavimas

Analoginio pobūdžio patikslinimus reikia daryti ir tada, kai komandoje JMP nurodytas šalutinis kreipinys , pvz. JMP [BX]; pagal nutylejimą assembleris nagrineja tokio pobudžio komandas kaip artimą šalutinį perdavimą.

Kas liečia atvejį, kai X nuoroda atgal, tai perdavimo tipą reikia būtinai nurodyti tik vienu atveju- esant tolimam tiesioginiam perdavimui: assembleris netgi žinodamas, kad X yra žymė iš kito komandinio segmeno, vis tiek nenagrinės tos komandos JMP X kaip tolimo perėjimo. Esmė tame, kad sutikdamas bet kokią žymę, assembleris automatiškai priskiria jai tipą **NEAR** (artimas) ir toliau vadovaujasi šiuo tipu. (assemblerio kalboje yra direktyva LABEL, kurios pagalba galima žymei priskirt tipą FAR). Pakeisti ši tipą galima tik su operatoriumi PTR.

MASYVAI, STRUKTŪROS

Asembleryje masyvai aprašomi su direktyvų, kurios aprašo duomenis, pagalba, naudojant DUP. Tačiau reikia tam tikrų paaikinimų.
Pvz. Masyvas X iš 30 elementų -žodžių aprašomas
X DW 30 dup(?)
Aprašant masyvą nurodomas elementų kiekis ir jų tipas, bet nenurodoma kaip jie numeruojami. Todėl tokiam aprašymui gali atitikti ir masyvas, kuriame elementai numeruojami nuo 0 iki 29 X(0,29), ir masyvas, kur numeruojami elementai nuo 1 iki 30 t.y. X(1,30) , ir masyvas su bet kuriuo kitu pradiniu indeksu k, t.y. X(k,k+29). Tokiu būdu programos autorius gali uždėti masyvui skirtingus indeksų kitimų diapazonus. Kokį diapason pasirinkti? Kartais indekso kitimo ribos gali būti griežtai apibrėžtos uždavinio sąlygoje (pvz. uždavinyje aiškiai pasakyta, kad elementai numeruojami nuo 1). Bet jeigu nenurodyta kaip tai turi būti daroma, tai geriau numeruoti nuo 0. Kodėl?
Pradėsime nuo to. kaip priklauso elemento adresas nuo to elemento indekso masyve. Tarkime mes nutarėme, kad elementai masyve numeruojami nuo k.
X db 30 DUP(?) ; X(k,29+k)
Tuomet teisinga yra, kad adresas (X(i))=X+2*(i-k) arba dar bendresniam lygyje, kai yra aiškus elementų ilgis (2), adresas
(X(i))=X+(type X)*(i-k)
Ta priklausomybė tampa daug paprastesnė, kai k=0 adresas
(X(i))=X+(type X)*I
Todėl paprastai ir skaičiuojama programuojant assemblerio kalboje, kad elementai masyve numeruojami nuo 0.
X dw 30 DUP(?) ; X(k,29+k) Ateityje taip ir darysime.
Daugiamačiams masyvams situacija analogiška.
Tegu turime dvimatįmasyvą (matricà) A, kuriame yra N eikuèiø ir M stulpeliø (N ir M konstantos) ir visi elementai - dvigubi žodžiai, o eilutės numeruojamos nuo k1, o stulpeliai nuo k2:
A DD N dup(M dup(?)) ; A [k1…N+(k1-1), k2…M+(k2-1)]
Čia mes manome, kad matricos elementai talpinami atmintyje eilutėmis: pirmosios M lastelių (dvigubų žodžių) uþima pirmos matricos eilutės elementai, sekantys M lasteliø - matricos antros eilutės elementai ir t.t. Žinoma matrisos elementus galima talpinti ir stulpeliais, bet tradiciškai priimta eilutinis elementų išdėstymas. Mes ją ir panagrinėsime. Tuo atveju matricos elemento adreso priklausomybė priklausomybė atrodys sekančiai:
adresas (A (I,j))=A+N*(type A)*(I-k1)+(type A)*(j-k2)
ir čia paprasčiausią vaizdą ši priklausomybė įgauna kai numeruojama nuo 0, t.y. k1=0,ir k2=0.
(A (i,j))=A+N*(type A)*i+(type A)*j

Kintamųjų su indeksu realizacija

Kitas klausimas , kuris iškyla dirbant su masyvais - tai kaip atliekamas priėjimas prie masyvo elementų, kaip organizuojami kintamieji su indeksu. Tam ir yra naudojama adresų modifikacija.

Adresų modifikacija

Iki šiol mes nagrinėjome komandas, kurių operandų adresai buvo nurodomi tiksliai,
pvz MOV CX,A
Tačiau bendru atveju komandoje kartu su adresu gali būti nurodytas ir kažkoks registras tarp kvadratinių skliaustų,
pvz, MOV CX,A[BX]
Tuomet komanda dirbs ne su nurodytu adresu A, o su taip vadinamu vykdomuoju adresu, kuris išskaičiuojamas pagal formulę:
Vykd. A=(A+[BX]) pagal mod 2^{16}, kur [BX] nurodo registro BX reikšmę. Kitaip sakant, prieš vykdant komandą, procesorius pridės prie adreso A, nurodyto komandoje, esamą registro BX reikšmę, gaus naują adresą ir būtent iš lastelės su tuo nauju adresu paims operandą. (Pati komanda tuo atveju nesikeičia, adresų išskaičiavimas atliekamas pačiam procesoriuje). Jei išskaičiuojant adresą gaunama per didelė suma, tai iš jos imami tik paskutiniai 16 bitų, ką ir rodo operacija pagal mod, nurodytoje formulėje. (Jei komandoje nenurodomas registras, tai Vykd. A laikomas lygiu adresui pačioje komandoje). Adreso pakeitimas komandoje į Vykdomąjį adresą vadinamas adreso modifikavimu, o registras , dalyvaujantis adresų modifikavime, vadinamas registru-modifikatoriumi.
Modifikatoriumi gali būti ne visi registrai, o tik vienas iš keturių: BX, BP, SI, DI.
Pvz. ADD A(SI),5
Modifikatorius šioje komandoje yra registras SI. Tegu dabar yra skaičius 100. Tada Vykd. A=A+(SI)=A+100. Vadinasi , pagal duotą komandą, skaičius 5 bus pridėtas prie lastelės, kurios adresas yra =A+100, o ne prie lastelės, kurios adresas yra A. Jeigu SI yra reikšmė -2 (0FFFEh), tada Vykd. A =A-2 ir todėl skaiėius 5 bus sudedamas su skaičiumi lastelėje, kurios adresas yra A-2.
Tai rodo, kad viena ir ta pati komanda gali dirbti su skirtingais adresais. Tai labai svarbu, dėl to ir yra adresų modifikacija. Vienas atvejas, kur yra naudojamas adresų modifikavimas, yra indeksavimas, naudojamas kintamųjų su indeksu realizavimui.
Indeksavimas
Išnagrinėsim pavyzdį:
Tegu turim masyvą X dw 100 dup(?) ; (0,…99)
ir reikia užrašyti į registrą AX jo elementų sumą.

Tam mums reikia pradžioje į AX užrašyti 0, o po to cikle atlikti operaciją AX:=AX+X(i), kai I keičiasi nuo 0 iki 99. Kadangi elemento X(I) adresas yra X+2*I, tai komanda, atitinkanti tą operaciją turi būti:
ADD AX,X+2*I, bet tokiakomanda negalima ir assembleryje ir mašininėj kalboj. Bet kokioj komandoj visi jos elementai, taip pat ir adresai, turi būti fiksuoti. Mūsų atveju adresas keičiasi priklausomai nuo I reikšmės. Vadinasi susidūrėme su problema: pagal algoritmą mūsų komanda turi dirbti su skirtingais adresais, o mašininės kalbos programavimo taisyklės leidžia tik fiksuotą adresą. Dėl tokio konflikto ir naudojamas adresų modifikavimas.
Išskaidysime kintamą adresą X+2*I į du :

- į pastovią dalį X, kuris nepriklauso nuo indekso I ir yra pastovus
- į kintamą dalį 2*I, kuri priklauso nuo indekso i.

Pastovi dalis užrašoma pačioje komandoje, o kintama dalis užrašoma į kokį nors registrą modifikatorių (pvz. Į SI), o jo pavadinimą taip pat užrašome į komandą, kaip modifikatorių:
ADD AX,X(SI), kur SI=2*I, o Vykd.A yra X+2*I
Kadangi registras modifikatoris yra vienas ir tas pats, tai tokiu būdu ši komanda turi fiksuotą vaizdą, t.y. patenkina mašininės kalbos taisykles. Iš kitos pusės, komanda dirba su Vykd.A, o jis gaunamas sudedant adresą X su registro SI reikšme (2*i), kuri gali keistis. Todėl , keisdami SI , mes priverčiam komandą dirbti su skirtingais adresais. Mums tik reikia pasirūpinti, kad registro SI reikšmė būtų teisingai keičiama. O tai jau yra visai paprasta: Pražioje į SI užkraunam 0, o po to didinam jo žingsnio reikšmę per du, rezultate mūsų komanda dirbs su adresais X, X+2, X+4 ir t.t.
Kadangi šiuo atveju modifikatorius naudojamas indekso saugojimui, tai toks registras vadinamas indeksiniu registru, o toks kintamojo su indeksu adreso išskaičiavimo būdas vadinamas indeksavimu.
Programos fragmentas atrodytų:

```
MOV AX,0          ;pradinė suma
Mov CX,100   ;ciklo skaitliukas
Mov SI,0          ;pradinė indekso reikšmė
L:
ADD AX,X(SI)      ;AX:=AX+X(I)
ADD SI,2          ;sekantis indeksas
LOOP L            ;ciklo tęsinys
```

Netiesioginiai kreipiniai
Išnagrinėsime dar vieną užduotą su adresų modifikavimu:
Tegu mums reikia atlikti užduotą:
Turim lastelę (vienas žodis), kurios adresas mums nežinomas, bet žinome, kad tas adresas yra uþduotas registre BX, ir mums reikia užrašyti sakykim reikšmę 300 į tą lastelę.
BX 

Jei mes iš anksto žinotume tos lastelės x adresą, tai mūsų užduotis būtų paprastai išsprendžiama t.y.
MOV x,300. Bet programos kūrimo metu mums tas adresas nežinomas, ir jo mes negalime nurodyti komandoje. Mums reikia vėl prisiminti, kad komandos dirba su Vykd.A , todėl mums reikia paimti tokį adresą ir tokį modifikatorių, kad sumoje jie mums duotų iš anksto nežinomą adresą. Lengva suprasti, kad mūsų atveju reikia paimti nulinį adresą ir registrą BX, kadangi tada Vykd.A =0+[BX]=0+x=x. Mūsų užduotis vykdoma
MOV [BX],300
Tokio adreso modifikavimo būdo ypatybė yra ta, kad kaip ir prieš tai, adresą pristatome kaip sumą dviejų komponentų, vienas iš kurių užrašomas komandoje, o kitas į registrą, bet jei prieš tai abu komponentai buvo ne nuliai, tai dabar vienas komponentas, kuri yra talpinamas komandoje, nulinis ir todėl visas adresas yra paslėptas registre. Gaunasi, kad

komandoje mes nurodom tik vietą (registrą), kur yra adresas. Toks adreso uždavimo būdas per tarpinę grandį, vadinamas netiesioginis adresavimas.
Taip pat pastebėsime, kad prie netiesioginio adresavimo, tenka patikslinti lastelės, į kurią kreipiamasi, ilgį.
Pvz,
MOV [BX],0 yra neteisinga, nes negalime nustatyti operandų dydžio, nes nulis gali būti 1 baitas, 2 baitai, o adresas užduotas registre BX gali būti adresas baito ar žodžio.
Patikslinam tai per operatorių PTR

```
MOV Byte PTR [BX],0        ;baito ilgio
MOV Word PTR [BX],0        ;žodis persiunčiamas
```

Modifikacija naudojant keletą registrų
Išnagrinėjom atvejus, kai adresų modifikacija atliekama naudojant vieną registrą-modifikatorių. Tačiau modifikavimo idėją lengva apibendrinti kelių modifikatorių atveju. Tam reikia komandose kartu su adresu nurodyti keletą tokių registrų. Leidžiama nurodyti iš kart du modifikatorius, kai vienas iš jų būtinai turi būti BX arba BP, o kitas SI arba DI(poromis modifikuoti BX iBP arba SI ir DI negalima). Galimas pavyzdys:

```
MOV AX,A[BX][SI]
```

Tuo atveju Vykd.A bus išskaičiuojamas naudojant formulę Vykd.A=(A+[BX]+[SI]) pagal mod 2^{16}
Modifikacija per du registruspaprastai naudojama, kai dirbame su dvimačiu masyvu. Pvz. Turime matricą A 10*20.

```
A DB 10 DUP(20 DUP(?))    ;A(0…   9,0…19)
```

ir reikia į registrą AL uþrašyti skaičių tokių eilučių, kurioje pradinis eilutės elemntas sutinkamas toje eiluteje dar kartą.
Išdėsčius matricos elementus OA eilutėmis (pirmieji 20 baitų -pradinė matricos eilutė, sekantys 20- antra matricos eilutė ir t.t.) elemento A(i,j) adresas bus A+20+i+j
Reikšmės 20*išsaugojimui išskirsime registrą BX, o dėl reikšmės j - registrą SI. Tada A[BX]- tai pradinis i-tosios matricos eilutės adresas, o A[BX][SI]- tai j-ojo elemento toje eilutėje adresas.

```
MOV AL,0     ;ieškomų elementų kiekis
;išorinis ciklas
MOV CX,10   ;Išorinio ciklo skaičius (eilučių sk.)
MOV BX,0              ;poslinkis nuo A iki 20+i
L:
MOV AH,A[BX]       ;pirmojo eilutės elemento reikšmė
MOV DX,CX ;išsaugojam skaitliuko turinį
;VIDINIS CIKLAS     ;
MOV CX,19   ; vidinis ciklas (pirmas elementas jau perimtas)
MOV SI,0              ;elemento indeksas viduje eilutės
L1:
INC SI          ;j:=j+1
CMP A[BX][SI],AH ;lyginam (I,j) elem. su pirmuoju el. eilutėje
LOOPNE L1   ;ciklas tæsiasi, kol yra nelygu, ir ne daugiau  19
JNE L2                ;jei nerandame pasikartojančių elementų eilutėje
```

```
INC AL              ;jei radom pasikartojantį elementą eilutėje
;  VIDINIO CIKLO PABAIGA
L2:
MOV CX,DX ;atstatome registro CX reikšmę dėl išorinio ciklo
ADD BX,20           ;pereinam prie kitos eilutės
LOOP L              ; kartojam ciklą 10 kartų
```

Modifikuojamų adresų užrašymas assemblerio programose

Patikslinsim adresų užrašymo taisykles:
Tegu A yra adresinė išraiška, o E bet kokia išraiška, tuomet assembleryje galimi trys adresų užrašymo formos komandose, kurie nurodo Vykd.A išskaičiavimo būdą:

A:	Vykd.A=A	
E[M]:	Vykd.A=(E+[M])	(M: BX,BP,SI,DI) MOD 2^{16}
E[M1][M2]	Vykd.A=(E+[M1]+[M2])	(M1: BX,BP M2:SI,DI)

Jei E=0, tai ją galima praleisti
Verta priminti kad modifikatoriais gali būti tik nurodyti keturi registrai (vienas iš jų), o jei modifikuojame ne pagal vieną modifikatorių, tai reikia nurodyti registrus poromis kaip parodyta anksčiau.
Registras BP naudojamas darbui su steku.
Taip pat reikia priminti, kad adresø modifikacija nekeièia adreso tipo.

Aptarsime kvadratinių skliaustų naudojimą aprašant komandos operandus:
1)

ĮVEDIMO / IŠVEDIMO KOMANDOS
Tai darbas su įvedimo – išvedimo irenginiais.
Įrenginiai skirstomi į

- vidinius: -Vidinė atmintis, procesorius;
- išorinius: visi kiti like.

Įvedimas ir išvedimas yra suprantami kaip apsikeitimas informacija tarp procesoriaus ir bet kurio kito įrenginio. Atskaitos taškas yra procesorius. Įvedimas yra procesas, kai duomenys iš išorinio įrenginio perduodami procesoriui. Išvedimas – kai duomenys iš procesoriaus perduodami išoriniams įrenginiams. Informacija tarp procesoriaus ir išorinių įrenginių perduodami per portus – specialius registrus. Visi portai yra vienbaičiai, du gretimai esantys portai gali būti nagrinejami kaip portas, kurio dydis yra žodis.
Potencialiai portai numeruojami nuo 0 iki ffff. Su kievienu išoriniu įrenginiu yra susiję konkretuas portas. Jų numeriai iš anksto žinomi. Išronis įrenginys gali užrašyti infomraciją arba nuskaityti ją iš porto.
Įvedimą atitiktų komandos IN AL,n arba IN AX,n.
Išvedimui yra komanda OUT n,AL arba OUT n,AX
Pagal komandą IN infomacija persiunčiame iš porto į registrą. Porto numeris ali būti užduotas skaičiais arba registre DX. Įvedimo/ išvedimo scenarijus panašus visiems įrenginiams ir vyksta per 2 portus. Vienas skirtas duomenims ir vadinamas duomenų

portu, kitas įvairiai valdymo iniformacjai ir vadinmas valdymo portu. Norint organizuoti įvedimą/ išvedimą reikia žinoti portų numerius, valdančiuošiuos ir atsakančiuosius signalus ir kitas detales. Skirtingiems įrenginiams ši informacija skirtinga, kiekvienoje programoje reikia iš naujo aprašyti. Kai centrinis procesorius nori pradėti informacijos apsikeitimą su Į/IŠ įrenginiais, tai jis į valdymo portą įrašo tam tikrą bitų kombinaciją, kuri yra traktuojama kaip pakvietimas ryšiui nustatyti. Jei įrenginys yra paruoštas darbui ir neužimtas, tai kaip atsakymas yra tame pačiame porte formuojama kita bitų kombinacija, kuri signalizuoja, kad įrenginys yra pasiruošęs apsikeitimui. Centrinis procesorius išlaukus tam tikrą laiką nuskaito informaciją iš minėto porto ir jei tai buvo informacija apie signalo nustatymą dėl apsikeitimo, pradeda informacijos skaitymą.

CP →Portas 1 išvesti informaciją
CP →Portas 2 užrašo išvedamą informaciją
Išvedimo įrenginys ←Portas 1
Išvedimo įrenginys ←Portas 2 siunčia signalą, kad užimtas įrenginys ir pradeda išvedimą.

Centrinis procesorius tuo pat metu pereina arba į laukimo būseną vis apklausdamas valdymo portą arba užsiima kitu darbu kol signalas “užimta” pasikeis iki bus baigtas išvedimas, tai išorinis įrenginys nusiūs į valdymo portą signalą apie sėkmingą darbo pabaigą arba apie klaidą. Tai aprašyti programose yra sudėtinga. Daugumoje programų naudojamos tos pačios įvedimo/ išvedimo operacijos. Tai įvertinant vieną kartą aprašomos dažnai naudojamos operacijos ir jos įjungiamos į operacinę sistemą, kuri visą laiką yra OA, kad jomis galėtų naudotis bet kokia kompiuteryje vykdoma programa,. Tai palengvina programuotojo darbą. Su portais tiesiogiai dirbame kai reikia kokios nors neįprastos operacijos.

Yra suprogramuota pertraukimo sistema, leidžianti apdoroti pertraukimus kiekvienam įrenginiui. Pertraukiant programos darbą turi būti įsimenami registrai CS ir IP. Jie įsimenami steke, tai pat išaugomos visos veliavėlių reikšmės.

Ekrano ir Klaviatūros operacijos per DOS

Paprastai programos turi pranešti apie savo darbo pabaigą, išvesdamos pranešimą ekrane. Taip pat programa gali išvesti įvairius informacinius pranešimus, pranešimus apie klaidas, gauti informaciją savo programai ir t.t.

Įvedimo ir išvedimo operacijoms atlikti naudojama komanda INT. Jos pagalba sistemai nurodome, kokias operacijas ir su kokiu įrenginiu reikia atlikti. Yra dvi INT komandos. Su komanda INT 10h valdymas perduodamas į BIOS. Su komanda INT 21h valdymas perduodamas pertraukimų aptarnavimo mechanizmui per DOS. Komanda INT nutraukia programos darbą ir perduoda valdymą į DOS, kad nustatyti kokį veiksmą reikia atlikti, jį atlieka ir po to vėl grąžina valdymą į programą. Labai dažnai tas pertraukimas vykdomas norint atlikti įvedimo ar išvedimo operacijas.

Bazinės DOS operacijos vekia bet kurioj aplinkoj.

Kursoriaus nustatymas

Ekraną galima įsivaizduoti kaip erdvę su koordinatėmis, į kurią galima nustatyti kursorių. Paprastas videoterminalas turi 25 eilutes (0-24) ir 80 stulpelių (0-79) .

Ekrano **vieta**	**koordinatės**			
	dešimtainėj sistemoj		šešioliktainėj sist.	
	Kolon.	Stulpelis	Kolon.	Stulpelis
Kairys viršutinis kamp.	00	00	00	00
Dešinys viršutinis kamp.	00	79	00	4Fh
Kairys apatinis kamp.	24	00	18	00
Dešinys apatinis k. kamp	24	79	18	4Fh
Ekrano centras	12	39/40 `	00	27/28h

Su komanda **INT 10h** galime kursorių pastatyti bet kurioje ekrano vietoje, o taip pat išvalyti ekraną.
Kursoriaus nustatymui naudojamas komandos kodas 2, kurį reikia patalpinti į registrą AH.
Ekrano eilutės numeris nurodomas registre DH,
Ekrano stulpelio numeris nurodomas registre DL,
Ekrano numeris nurodomas registre BH. Paprastai tai yra 0
Pvz. Kursoriaus nustatymas į 5 eilutę ir 12 kolonėlę atliekamas per komandų seką:

```
Mov AH,02
Mov BH,00
Mov DH,05
Mov DL,12
INT 10h
```

Ekrano valymas

Dėl aiškumo pradedant vykdyti programą , kad būtų aiškūs užklausimai ir rezultato formavimai, galima išvalyti ekraną. Ekrano sritis gali būti išvaloma nuo bet kurios pozicijos ir apimti bet kokią sritį. Pradinis adresas turi būti mažesnis, negu antrasis adresas.

- Pradinės ekrano srities adresas užduodamas registre CX,
- antras adresas nurodomas registre DX,
- BH turi būti reikšmė 07
- AX turi būti reikšmė 0600h

```
MOV AX,0600h      operacijos kodas- ekrano valymas
MOV BH,07
MOV CX,0000             eilutė ir stulpelis
MOV DX,184Fh      eilutė ir stulpelis
INT 10h
```

Informacijos išvedimas į ekraną

- turi būti suformuotas pranešimas duomenų segmente, kuris turi baigtis “$” ženklu.
- Registro AH reikmė=09h
- naudojama komanda INT 21h
- Registre DX turi būti nurodytas pranešimo adresas (LEA DX,pran).

Jei netyčia bus pamirštas “$” ženklas, tai į ekraną bus išvedami visi simboliai nuo nurodyto adreso, kol bus sutiktas šis “$” ženklas OA.

Ekrano valdymui priskiriamos komandos taip pat ir

□	kursoriaus pervedimas į kitą eilutę LF	10	0Ah
□	kursoriaus nustatymas į eilutės pradžią CR	13	0Dh

Paprastai tie du kodai eina greta, kad tvarkingai atspausdinti pranešimus ekrane. Juos galima aprašyti bet kurioj programos dalyje naudojant direktyvą EQU. Kaip žinia ji nekeičia nei ilgio programos nei struktūros.

```
CR      EQU   13
LF      EQU  10
```

Duomenų įvedimas iš klaviatūros

Duomenų įvedimo procedūra iš klaviatūros yra paprastesnė, nei išvedimas į ekraną. Reikia tam tikrų paramertų, aprašančių įvedimo procedūrą, naudojant komandą INT:

□ turi būti nustatytas maksimalus įvedamų simbolių skaičius. Tai būtina, kad būtų galima perspėti vartotoją apie per ilgą pranešimą. Kiti nenumatyti simboliai, įvedami virš limito, tiesiog ignoruojami. Maksimalus įvedamų simbolių skaičius vienos komandos metu yra 255 simboliai.

□ turi būti išskirtas laukas, kur komanda baigusi duomenų įvedimą, įrašys tikrąjį įvestų simbolių skaičių

□ reikia nurodyti lauko adresą, kur turi būti talpinami įvedami simboliai. Tas laukas turi būti aprašytas duomenų segmente.

Duomenų įvedimui naudojame operaciją, kurios kodas 0Ah, ir jis nurodomas registre AH o parametrų lauko adresas turi būti užduotas registre DX, t.y. atlikti įvedimą galime tokiu būdu:

```
□   MOV AH,0Ah          operacijos kodas
□   LEA DX,IVSRIT       parametrų ir įvedimo sritis.
□   INT 21h              įvedimo vykdymas
```

Parametrų ir duomenų ivedimo sritis turi būti aprašyta duomenų segmente, tiksliai nurodant pirmą baitą, kuris naudojamas maksimaliai galimų įvedamų simbolių skaičiui nustatyti, pvz.

```
IVSRIT      DB    25      ;Maksimaliai galima bus įvesti 25 simbolius
            DB    ?       ;Čia gausime realiai įvestų simbolių skaičių
            DB    25 DUP (‘ ‘) ;Čia paklius įvesti iš klaviatūros simboliai.
```

Įvedimas visada baigiamas paspaudus klavišą “ENTER”- jo kodas 0Dh. Šis kodas taip pat formuojamas duomenų įvedimo srityje, tačiau jis neįskaitomas į įvestų simbolių skaičių. Kadangi šis simbolis yra formuojamas įvedimo srityje, tai programuotojas turi žinoti, kad realiai įvedamų simbolių skaičius bus 1 mažesnis už nurodytą parametrų srityje. Įvesti simboliai keičia įvedimo srities turinį duomenų segmente. Dirbant cikle, dėl duomenų korektiškumo, reikia prieš įvedant naujus duomenis, tą sritį išvalyti.

```
MOV   CX,25
LEA   BX,ivsrit+2
 cikl:
        MOV [BX],20h
```

```
        INC   BX
        LOOP cikl
```

PVZ.
Parašyti programą, kuri atspausdintų ekrane programos autoriaus vardą. Autorius savo vardą įveda iš klaviatūros. Pradedant darbą ekranas turi būti išvalytas. Autorius savo vardą spausdina toje ekrano eilutėje, kuri atitinka jo įvestų simbolių skaičiui eilutės viduryje.

Programa

Operacija 08H leidžia įvesti vieną simbolį, kuris formauojamas registre AL. Šiuo atveju įvestas simbolis nerodomas ekrane.

Darbas su diskiniais įrenginiais

Įrašymui į diskus ir skaitymui iš jų naudojami kodai:
3Ch sukurti failą
3Dh Atidaryti failą
3Eh Uždaryti failą
3Fh perskaityti failą
40h Užrašyti įrašą į failą
2H Perstumti failo nuorodą

Dirbant su įvairiomis diskinėmis funkcijomis, reikia sistemai iš anksto pranešti eilutės, kuriame yra failo adresas: t.y. diską, kelią iki to failo diske ir pilną failo vardą, adresą.Bet kurį iš tų duomenų galima praleisti, aprašant ten 0. Eilutė užrašoma tarp apostrafų. Tos eilutės pabaiga nurodoma per 00h. Maksimalus eilutės ilgis yra 128 baitai.
Pvz.
Pran1 Db ‘C:A17RAND.ASM’,00h

Pran2 Db ‘C:\UTILITY\A17RANDM.ASM’,00h papildomai nurodome ir kelią iki failo.

Failų deskriptoriai
Kai kuriuos deskriptorius galima panaudoti tiesiogiai. Tai standartinių įrenginių deskriptoriai:
- įvedimas
- Išvedimas
- pranešimų apie klaidas išvedimas
- pagalbinis įrenginys
- printeris

Kiti įvedimo/išvedimo įrenginiai, dirbantys su failais, reikalauja failų deskriptoriaus. Tai reikia paprašyti tokio deskriptoriaus iš sistemos. Todėl failas diske pirmiausia turi būti atidarytas. Skirtingai nuo duomenų perdavimo displėjui ir priėmimo iš klaviatūros,

duomenų apsikeitimui su diskais, sistema turi kreiptis į FAT ir direktorijų struktūrą, kad gauti priėjimą prie failo.. Įrašai FAT ir direktyvų struktūroje turi atsinaujinti sutinkamai su pasikeitimais file. Kiekvienam failui į kurį kreipiasi programa darbo metu, turi būti sukurtas unikalus deskriptorius. Sistema priskiria failui deskriptorių jo atidarymo metu failo skaitymui arba naujo failo sukūrimui dėl informacijos išvedimo. . Šias operacijas naudoja funkcijos 3Ch ir 3Dh per pertraukimą INT 21h. Failo deskriptorius – tai unikalus šešioliktainis skaičius, kuris grąžinamas į registrą AX ir saugomas atmintyje žodžio formate. Deskriptorius naudojamas visose vėlesnėse įvedimo/išvedimo operacijose, dirbant su failu. Paprastai pirmas atidaromas failas turi deskriptorių 05, antras –06 ir t.t. Programos segmento prefiksas saugo deskriptorių lentelę, kuri pristatoma pagal nutylėjimą ir saugo joje 20 deskriptorių- vienu metu galimų atidarytų failų skaičių. . Galima tą sritį praplėsti, naudojant funkciją 67h ir pertraukimą INT 21h

Grąžinamų klaidų kodai
Operacijos, kurios naudoja deskriptorius, grąžina veiksmo vykdymo būseną per vėlevėlę CF ir registrą AX. Operacijai pasibaigus gerai, CF reikšmė yra 0 ir yra atliekama užsakyta operacija. Esant nesėkmei, CF nustatoma 1 , o AX talpinamas klaidos kodas, priklausantis nuo operacijos. Tai yra kodai intervale tarp 01 ir 36.

Failiniai nurodymai
Sistema palaiko atskirą failinę nuorodą kiekvienam failui su kuriuo dirba programa. Sukūrimo ir atidarymo operacijos inicializuoja failinę nuorodą, nustatydama jį ir jis rodo į failo pradžią. Bendrai jis rodo poziciją faile su atatinkamu poslinkiu jame. Kiekviena skaitymo/rašymo operacija priverčia sistemą padidinti nuorodos reikšmę per skaičių, sutinkamai su perduotų baitų skaičiumi. Tuomet failinė nuoroda rodo į vietą faile, nuo kurios turi būti pradėtas rašymas (kito įrašo). Nuorodos naudojamos nuosekliam priėjimui ir laisvam priėjimui prie failo. Laisvam priėjimui prie norimo įrašo faile galima naudoti komandą 42H per pertraukimų sistemą INT 21h

Failų kūrimas
Jiems kurti naudojama procedūra, sudroma iš žingsnių:
- Panaudoti elutę, kad iš sistemos gauti failo deskriptorių
- Iškviesti funkciją 3CH per pertraukimo procedūrą INT 21h, kad sukurti įrašą apie failą direktorijų struktūroje.
- Iškviesti funkciją 40H per INT 21h kad užrašyti į failą atskirus įrašus.
- Baigus darbą su failais iškviest funkciją 3EH per INT 21h kad uždaryti failą.

Funkcija 3CH
Ši funkcija skirta naujam failui sukurti arba perrašyti seną failą su tuo pačiu vardu. Tam reikia į registrą CX užrašyti kuriamo failo atributus, o į registrą DX reikia užrašyti eilutės adresą. Šioje eilutėje yra nurodomas kelias kuriamam failui sistemoje.
PVZ. Reikia diske C sukurti naują failą be atributų

```
PATHNAME1       DB      'C:\ACCOUNTS.FIL',00h
FILHAND1        DW      ?       ;failo deskriptorius


                MOV AH,3CH          ; paprašyti sukurti failą
                MOV  CX,00          ;nenaudojame atributų
                LEA  DX,PATHNAM1    ;kelio eilutės adresas
                INT  21h            ; pertraukimas operacijai įvykdyti
                JC   klaida  ; jei įvyko klaida
                MOV  FILHAND1,AX    ;įsiminti failo deskriptorių
```

Esant teigiamam rezultatui, sistema sukurs įrašą direktyvų struktūroje su užduotais atributais , nustatys vėlevėlės reikšmę CF 0 ir į AX patalpins failo deskriptorių.
Šis deskriptorius bus naudojamas programos darbe dirbant su failu.. Atidaromas failas ir jo nuoroda tampa lygi 0. Dabar jau galima rašyti į failą. Jei failas su nurodytu vardu jau yra sukurtas, tai nuoroda bus nustatoma lygi 0, ir nauji duomenys bus rašomi ant senųjų viršaus. Jei pasitaikys klaida , įjungiama vėlevėlė CF ir į AX yra patalpinamas klaidos kodas.

Funkcija 40H
Ji naudojama kai reikia išsaugoti įrašus diskiniam faile. Į registrą BX reikia patalpinti failo deskriptorių (į kurį mums reikia įrašyti duomenis), į registrą CX įrašomų baitų kiekį, o į DX išvedamų duomenų adresą.

PVZ. Į failą ACCOUNTS.FIL užrašome 256 baitus iš srities DSKAREA:

```
FILHAND1    DW    ?       failo deskriptorius
DSKAREA     DB    256 Dup ('*')  ;išvedimo sritis


MOV   AH,40H            ; paprašyti užrašyti duomenis į failą
MOV   BX,FILHAND1       ;Įrašyti failo deskriptorių
MOV   CX,100h           ;patalpinti įrašomo fragmento ilgį
LEA   DX,DSKAREA        ;patalpinti išvedimo srities adresą
INT 21h                 ;įvykdyti pertaukimo operaciją duomenų rašymui
JC    klaida  ;         Jei buvo įrašymo klaida
CMP   AX,100h           ; Ar visi baitai įrašyti
JNE   klaida1           ;Jei ne visi baitai įrašyti, eiti į klaidos apdorojimą.
```

Korektiškai, operacija užrašo visus nurodytus baitus, padidina failo nuorodą, nustato CF vėlevėlės reikšmę 0 ir į AX užrašo tikrai įrašytų baitų skaičių. Disko perpildymas gali iššaukti klaidą, kad ne visi baitai bus užrašyti, todėlprogramoje reikia numatyti klaidos galimybę. Nekorektiškai atlikta operacija nustato CF reikšmę 1ir į AX patalpina klaidos kodą

Funkcija 3EH

Ši funkcija skirta failo uždarymui. Baigus įrašymą, reikia failą uždaryti per šią funkciją, įvykdant atatinkamą pertraukimą. Tam BX reikia nurodyti uždaromo failo deskriptorių ir iškviesti pertaukimo programą.

```
MOV   AH,3EH          ;Paprašyti uždaryti failą
MOV   BX,FILHAND1     ;Užrašyti failo deskriptorių
INT   21h             ;Įvykdyti pertraukimą
JC    klaida          ;perduoti valdymą klaidos apdorojimui.
```

Paruošti programą
Įvesti iš klaviatūros duomenis ir juos užrašyti į failą.

Failo skaitymas iš disko

Šis procesas apima procedūras

- Eilutės panaudojimas failo deskriptoriaus gavimui iš sistemos
- Funkcijos 3DH panaudojimas failo atidarymui
- Funkcijos 3FH panaudojimas duomenų nuskaitymui iš disko
- Funkcijos 3EH panauojimas failo uždarymui

Funkcija 3DH
Kad gauti duomenis iš failo, programa turi atidaryti failą. Tam naudojame funkciją 3DH ir jos pagalba per INT 21h atidarome failą. Funkcija tikrina ar toks failas yra, ir jei jis yra, nustato jam deskriptorių. Užkraukite eilutės adresą registre DX, o registre AL patalpinkite 8 bitų priėjimo prie duomenų kodą.

Bitai	paklausimas	Bitai	paklausimas
000.	tik skaitymui	1-	Rezervuota
001	tik rašymui	4-6	Režimas
010	skaitymui ir rašymui	7	Palikimas

Prieš skaitymą iš failo programa turi panaudoti funkciją 3Dh dėl failo atdarymo, o ne 3EH dėl jo sukūrimo.

```
PATHNAM1  DB     'C:\ACCOUNTS.FIL'00H
FILEHAND  DW     ?        ;failo deskriptorius

          MOV   AH,3DH
          MOV   AL,00
```

```
        LEA   DX,PATHNAM1
        INT   21h
        JC    klaida
        MOV   FILEHAND,AX
```

Jei failas su nurodytu vardu yra, tai operacija nustato įrašo ilgį 1, priima esamus failo atributus, nustato failo nuorodą 0, nustato CF reikšmę 0 ir į AX patalpia atidaryto failo deskriptorių.Jis bus naudojamas programoje visą laiką iki pat failo uždarymo.
Jei failo nėra, tai CF nustatomas 1 ir į AX patalpinamas klaidos kodas 02, 03, 04, 05 arba 12 . Būtina tikrinti vėlevėlės CF kodą. Pvz. atidarant failą jam gali būti priskirtas deskriptorius 05 ir jį lengva supainioti su klaidos kodu 05 – kas reiškia priėjimas prie failo negalimas.

Funkcija 3FH
Ji skirta informacijos skaitymui iš failo diske. Reikia failo deskriptorių patalpinti registre BX,baitų skaičių, kurį reikia perskaityti, patalpinti į CX, o įvedimo srities adresą į registrą DX.

```
FILEHAND  DW    ?             ; failo deskriptorius
INAREA    DB    512 DUP(?)    ;įvedimo sritis
        MOV   AH,3FH          ;paprašyti skaityti failą
        MOV   BX,FILEHAND     ;patalpinti failo deskriptorių
        MOV   CX,200h         ;Patalpinti skaitomų baitų kiekį
        LEA   DX,INAREA       ; patalpinti įvedimo srities adresą
        JC    klaida
        CMP   AX,00           ;Ar visi baitai nuskaityti
        JE    pabaiga         ;Jei failas perskaitytas.
```

Jei operacija atlikta korektiškai, tai programai perduodami duomenys, nustatoma CF reikšmė 0 ir AX patalpina tikrą nuskaitytų baitų kiekį. Pavyzdyje AX reikšmė 0 reiškia kad buvo bandymas skaityti iš jau perskaityto failo. Tai perspėjimas o ne klaida. Nekorektiškas operacijos atlikimas nustato CF 1, o į AX patalpina klaidos kodą 05 arba 06.
Kadangi atidarytų vienu metu failų kiekis yra ribotas, programa paeiliui kreipdamasi į eilę failų turi uždaryti bet kurį iš jų , kai tik darbas su juo baigtas.

DARBAS SU REALIAIS SKAIČIAIS

Realių skaičių aprašymas

Assemblerio programoje dešimtainius skaičius galime pateikti specialiu realių skaičių formatu, kurį pažysta koprocesorius. Tai yra skaičius sudarytas iš mantisės ir laipsnio. Mantisė yra ženklas, sveikoji dalis ir trupmeninė dalis. Pvz 234.56E+02 turės mantisę 234.56, o laipsnis bus +02

Pvz. 2.5

+0.4646

1.024E+04; $1.042*10^4$

Duomenis galima aprašyti direktyvos DD pagalba- t.y. dvigubas žodis, kuris nurodo assembleriui, kad duomenims saugoti yra skirta 32 skirsniai. Realus skaičius, saugomas kaip dvigubas žodis, vadinamas trumpu realiu skaičiumi. Skaičių galima aprašyti kaip šešioliktainį ar kaip dešimtainį.

Dešimtainiai skaičiai gali turėti tikslumą iki 8 ženklų po kablelio.

```
Pvz.    DD      12345.678
        DD      1.5E+02         1.500
        DD      2.56E+38 ir tt
```

Direktyva DQ aprašo keturgubą sžodis ir skiria vietą 8 baitų kintamajam. Ši direktyva analogiška direktyvai DD. Naudojama C,C++ ir Visual Basic kalbose.

```
Float1  DQ      1.5E+10
Float2  DQ      2.56E+307
        DT      darbiniai skaičiai. Užima 10 baitų. Duomenys gali būti saugomi kaip
```

supakuoti realūs skaičiai, sveiki dešimtainiai skaičiai, supakuoti dešimtainiai skaičiai arba dvejetainiai skaaičiai. Pats skaičius gali būti teigiamas arba neigiamas ir sudarytas iš 18 skaitmenų.

```
PAK-1 DT        012345678912345678
```

Assembleris traktuoja, kad laipsnio rodiklis išreikštas šešioliktainiu skaičiumi arba gali būti panaudotas indikatoriaus dešimtainės reikšmės.

Laikini realūs skaičiai gali būti dešimtainiai ar šešioliktainiai skaičiai. Šešioliktainis skaičius gali būti sudarytas iš 20 skirsnių.

```
FLOAT1          DT      1.5
```

Koprocesorius yra kaip antras procesorius, dirbantis lygiagrečiai pagrindiniam procesoriui. Matematinis koprocesorius8087 skirtas 8088 efektyvumui padidinti. Koprocesorius kartais keičiamas bloku darbui su duomenimis su plaukiojančiu kableliu. Yra komandų sistema darbui su koprocesorium. Ji leidžia dirbti su sveikais skaičiais ir su realiais skaičiais. Visa darbo logika realizuota aparatūriškai.

Koprocesoriaus architektūra

Yra aštuoni atskirai adresuojami registrai. Jie surikiuoti kaip stekas ir turi vardus. Steko viršūnė yra **ST**

Registrai yra **ST(0), ST(1), ST(2), ST(3), ST(4), ST(5), ST(6), ST(7),** Atatinkamai tai yra **R0-R7.**

Į registrą **ST(0)** dažniausiai kreipiamasi kaip į **ST** ir jis yra steko viršūnėje.

Skaičiai saugomi registruose yra naudojami skaičiavimams ir turi laikiną realių skaičių formatą, kurių ilgis yra 10 baitų.
Kai koprocesorius saugo rezultatą po aritmetinių operacijų atmintyje, jis perveda skaičius iš laikino formato į vieną iš nustatytų: sveikų , skaičius, trumpo realaus , ilgo realaus. Skaičiaus perdavimas iš koprocesoriaus vyksta per atmintį, todėl pirmiausia kol škviesime koprocesorių, reikia išsaugoti reikiamus operandus atmintyje. Tada koprocesorius paima operandus iš atminties ir saugo juos steko registruose. Atlikus skaičiavimus, koprocesorius patalpina rezultatą atmintyje ir signalizuoja pagrindiniam procesoriui, kad užduotis įvykdyta.

Valdantys registrai
Yra penki koprocesoriaus valdantys registrai. Trys iš jų 16 skirsnių. Tai valdantis žodis, būsenos žodis, antraštės žodis, t.y. control word, status word ir tag word.
Kiti du registrai yra 32 skirsnių. Tai yra komandų nuorodos ir operandų nuorodos.

Valdantys registrai turi savo vardus
SR būsenos žodis
CR Valdantis žodis
TW antraštės žodis
FIP saugo paskutinės įvykdytos komandos adresą
FDP saugo paskutinės įvykdytos komandos operando adresą.

Koprocesoriaus Komandų formatai
Visos koprocesoriaus komandos prasideda raide **F.** pagal tai koprocesorius atpažysta kad tai jo komanda. Antra raidė komandoje gali būti **B arba I** ir nustato operandų formatą.
Ridė B nurodo kad tai yra dvejetainis dešimtainis skaičius
Raidė I nurodo sveiką skaičių. Jei nurodytų raidžių komandoje nėra, tai skaitoma, kad operandas yra realus skaičius.
FBLD yra dvejetainis dešimtainis skaičius
FILD yra sveiki skaičiai
FMULrealių skaičių daugyba
Yra šeši komandų formatai, pateikti lentelėje.

Komandos formatas	Komandos kodas	Operandai (priėmėjas / šaltinis)	Pavyzdžiai
Klasikinis stekas	Fop	{ST(1),ST}	FADD
Klasikinis stekas išėmėjas	Fop	{ST(1),ST}	FSUBP
Registrai	Fop	ST(n),ST ST,ST(n)	FMUL ST(1)ST FDIV ST,ST(3)
Registrai išėmėjas	FopP	ST(n),ST	
Reali atmintis	Fop	{ST},memReal	
Sveikoji atmintis	Flop	{ST},memInt	

Operanduose n nurodo registro numerį.

MemReal nurodo 16 skirsnių sveiką skaičių.memReal gali būti trumpas realus skaičius (4baitai) ilgas realus skaičius (8baitai()ir supakuoti BCD(10baitų) ir darbiniai realūs(10baitų)
MemInt gali būti žodis (2baitai), trumpas sveikas skaičius (4 baitai), ilgas sveikas skaičius (8baitai).

Op nurodo aritmetinę operaciją.
Operandai figūriniuose skliaustuose yra operandai ir tiksliai nekoduojami, bet laikomi dalimi operacijos. Op gali būti
ADD sudėti operandą šaltinį su operandu priėmėju
SUB Atimti operandą šaltinį iš operando priėmėjo
SUBR Atimti operandą priėmėją iš operando šaltinio
MUL sudauginti operandą šaltinį su operandu priėmėju
DIV padalinti operandą priėmėją iš operando šaltinio
DIVR padalinti operandą šaltinį iš operando priėmėjo

Koprocesoriaus komandų operandai
Koprocesoriaus komandos gali turėti iki dviejų operandų, bet vienas I jų turi būti koprocesoriaus registras.Negalima naudoti registrų AX ir BX. Taip pat negalima atlikti operacijų su duomenimis **atmintis- atmintis.**

Klasikinis stekas
Komandos valdo registrus steko vitšūnėje. Nereikia jokių operandų. Pagal nutylėjimą ST yra siuntėjas (šaltinis), o ST(1) –operandas priėmėjas. Rezultatas laikinai saugomas ST(1). Po to ST išimamas iš steko , palikdamas rezultatą steko viršūnėje.

Pvz. FADD	;ST(1)=ST		
	Iki	Po operacijos	
ST	100.00	ST	120.00
ST(10	20.00	ST(1)	

Realūs ir sveiki operandai atmintyje

Komandos turi vieną įsivaizduojamą operandą ST. Antras operandas nurodomas tiksliai. Gali būti sveikas arba realus skaičius atmintyje.

FADD	mysingl	; ST=ST+mysingle
FSUB	mysingle	;ST=ST-mysingle
FSUBR	mysingle	;ST=mysingle-ST

Darbui su sveikais skaičiais atmintyje komandos transformuojamos į komandas FIADD, FISUB, FISUBR

Koprocesoriaus registrai
Komandos darbui su registrais naudoja koprocesoriaus registrus kaip paprastus. Vienas iš operandų yra būtinai ST

```
FADD ST,ST(1)       ST=ST+st(1)
FDIV  ST,ST(3)      ST=ST(3)/ST
FMUL ST(2),ST       ST(2) =ST(2)*ST
```

Išėmimas iš registrų

Komandos darbui su registrais yra analogiškos komandoms darbui su registrais, bet su viena išimtim – įvykdžius komandą, registras ST išimamas iš steko.

FADDP sumuoja registrus ST ir ST(1), o rezultatą formuoja registre ST(1). Po to ST išimamas iš steko, o ST(1) pesiunčiamas į ST.

FADDP st(1),ST

	Iki operacijos	Proceso metu	Po operacijos
ST	200.00	200.00	232.00
ST(1)	32.00	232.00	

Pvz. 6*2+5

Veiksmai
Paimti operandus ir patalpinti steke
Nuskaičius operatoriųiš įėjimo paimti du operandusiš steko, įvykdyti operaciją ir patalpinti rezultatą steke

Pavyzdžiai (Užduotys.)
Paprastosiso programos
Darbas su iš anksto aprštytais duomenimis

1. Apskaičiuokite pagl formulę y=a+b-c+1 reikšmę, kai a=17, b=-3, c=2;

Duomenys a,b,c aprašyti duomenų segmente.

2. Aprašome 10 baitų ilgio tekstą duomenų segmente. Reikia sukeisti vietomis tame lauke 1 ir 5 baitus. Į 3 baitą pasiūsti simbolį “E” , į 6 ir 7 baitus nusiūsti reikšmes 78.

Šakotosios programos
1. Sudaryti programą funkcijos skaičiavimui
F=2a/(x+y), kai a>10
F=x-y kai a<10
a, x ir y reikia įvesti iš klaviaturos. Įvedimui naudoti paprogramę. Rezultatą atspausdinti ekrane.

Ciklai
Peradresavimo ciklai

1. Turime paeiliui masyve įrašytus šimtą skaičių. Reikia rasti šių skaičių sumą.

2. Sudaryti programą, kuri suskaičiuotų kiek yra raidžių “A” masyve T1 ir kiek yra raidžių “B” masyve T2. Masyvų pabaigos žymimos simboliu “*”.

Ciklai cikle

3. Turime masyvą, kurio struktūra
Grupes Numeris
Studento Pavarde
Egzaminų Pažymiai
Pažymių Vidurkis

Reikia suskaičiuoti bendrą vidurkį. Čia nežinomas studentų skaičius. Jų kiekį reikia nustatyti pagal masyvo pabaigą. Masyvo pabaiga yra ‘***’

Darbas su realiais skaičiais
Indenų uždavinys
1627 metais indėnai pardavė Manheteno salą už 24 dolerius. Kiek jie turetų pinigų šiais metais, jei šiuos dolerius būtų padeję į banką ir bankas moketų 3 procentų metines palūkanas.
Jei pradinė suma yra **S0** ir kasmet ji padideja **p** procentų, tai po **n** metų gaunama suma **Sn** apskaičiuojama iš formulės
Sn=S0(1+p)n
Šiuo atyveju S0=24, n=2007-1627=380, p=0.03.
Sumos skaičiavimo algoritmas
1) Skaitliukas =n t.y. kiek metų buvo skaičiuojamos palūkanos
Galutinė suma S metų gale
2) S=S0 t.y Suma nuo kurios skaičiuojamos palūkanos, kiekvienais metais ši suma yra padidėjusi ankstesnių metų atžvilgiu suskaičiuotomis palūkanomis.
3) S=S(1+p) T.y. suskaičiuojame sumą su palūkanomis ir ji bus išeities skaičius kitų metų skaičiavimams (aišku jei indelininkas neatsiims šių palūkanų ar visų pinigų). Šiam punktui vykdyti organizuojame ciklą tiek kartų kiek metų reikia skaičiuoti palūkanas.

Atsiskaitymui pateikti programo listingą
Programas tureti kompiuterinėje laikmenoje, kad būtų galima pademonstruoti jų darbą. Vykdymo metu programa turi pradėti darbą nuo autoriaus pavardės ir grupės numerio pateikimo displėjaus ekrane.

Pavyzdys paprastos programos
Jos vykdymui reikia pirmiausia paruošti failą klasikinę schemą

I paruošiame programos išeities tekstą ir jį užrašome į filą, patikslinant failo tipą **.ASM**

```
 ;-steko segmentas-
stekas segment stack
db 256 dup (?)
stekas ends
;-duomenu segmentas-
duom segment
a dw 17 ;
b dw -5 ; | - priskiriami duomenys kintamiesiems
c dw 2 ;
ten db 10
ats db 4 dup (?),'$'
aut db 'Kursinio darbo uzduotis Nr.1',13,10
db 'Atliko Vardas Pavardele, II-0X',13,10
db '2007 07 07',13,10,'$'
abc db 13,10,'a = 17'
db 13,10,'b = -5'
db 13,10,'c = 2',13,10,'$'
txt db 'a+b-c+1=  ',7,'$'
minus  db '-'
duom ends
;-programos segmentas-
program segment
assume cs:program, ds:duom, ss:stekas
start:
mov ax,duom ;
mov ds,ax ; Į ds segmentinį registrą užkrauname duomenųsegmento adresą
;Išvalome ekraną
mov ax, 0002h
int 10h
;Išvedame praneėšimus ekrane
mov ah,09h
lea dx,aut        ;pirmo pranešimo adresas
int 21h
mov ah, 09h
lea dx, abc       ;antro pranešimo adresas
int 21h
mov ah, 09h
lea dx, txt       ;trečio pranešimo adresas
int 21h
;Atliekame veiksmus
xor ax,ax         ;išvalome al registrą
```

```
mov ax,a        ;al:=a   , nusiunčiame a reikšmę į registrą
add ax,b        ; al:=a+b Sudedame skaičius
sub ax,c        ; al:=a+b-c atimame c reikšmę
add ax,1        ; al:=a+b-c+1, pridedame vienetą
;Tikrinamar rezultats teigiamas
CMP ax,0
JL      neig
Teig:
lea si, ats+3   ;patalpiname atsakymo adresą
mov ah,0
mov cx, 3       ;Tai darome, kad normaliai isvestu atsakyma
ciklas:         ;paruošiame rezultatą išvedimui, suformuodami ASCI kode
div ten         ;daliname iš dešimt
add ah, 30h     ;
mov [si],ah     ;masyvo skaitliukas
dec si          ;mažinam skaitliukąa vienetu
mov ah, 0       ;išvalom ah:=0
loop ciklas     ;grįžtame į ciklo pradžią
mov ah, 09h
lea dx, ats     ;atspausdina atsakymą
int 21h
;-grązžna į DOS
mov ah,4ch
int 21h
Neig:
MOV ats,minus
JMP teig
program ends
end start
```

II sutransliuojame programą

III Sulinkuojame programą

2 paprastos programos pavyzdys ir cilkas

```
;-steko segmentas-
stekas segment stack
db 256 dup (?)
stekas ends
;-duomenu segmentas-
duom segment
a db 78          ;
b db –“E”        ; | - priskiriami duomenys kintamiesiems
masyvas db “1a2bcDHIJK”
aut db 'Darbo uzduotis Nr.2',13,10
db 'Atliko Vardas Pavardele, II-0X',13,10
db '2007 07 07',13,10,'$'
duom ends
;-programos segmentas-
program segment
assume cs:program, ds:duom, ss:stekas
start:
mov ax,duom ;
mov ds,ax ; Į ds segmentinį registrą užkrauname duomenųsegmento adresą
;Išvalome ekraną
mov ax, 0002h
int 10h
;Išvedame praneėšimus ekrane
mov ah,09h
lea dx,aut       ;pirmo pranešimo adresas
int 21h
mov ah, 09h
lea dx, abc      ;antro pranešimo adresas
int 21h
mov ah, 09h
lea dx, masyvas          ;trečio pranešimo adresas
int 21h
;Atliekame veiksmus
LEA    SI,masyvas
MOV   AL,0(SI)
MOV   BL,4(SI)
MOV   0(SI),BL
MOV   4(SI),AL
MOV   AL,a
MOV   2(SI),AL
MOV   AL,b
MOV   AH,b
MOV   5(SI),AX
Masyvas po pakeitimu
```

```
MOV AH,09H
LEA DX,mayvas
INT 21H
;-grąžžna į DOS
mov ah,4ch
int 21h
program ends
end start
```

3 pavyzdys Šakotosios programos ir ciklas

```
;-steko segmentas-
stekas segment stack
db 256 dup (?)
stekas ends
;-duomenu segmentas-
duom segment
a db 17 ;
x db 5 ; | - priskiriami duomenys kintamiesiems
y db 2 ;
ten db 10
ats db 5 dup (?),'$'
aut db 'Darbo uzduotis Nr.3',13,10
db 'Atliko Vardas Pavardele, II-0X',13,10
db '2007 07 07',13,10,'$'
abc db 13,10,'a = 17'
F1      db 13,10,’F = 2a/(x+y)’,13,10,'$'
F2      db ‘F=(x-y)  ',7,'$'
duom ends
;-programos segmentas-
program segment
assume cs:program, ds:duom, ss:stekas
start:
mov ax,duom ;
mov ds,ax ; Į ds segmentinį registrą užkrauname duomenųsegmento adresą
;Išvalome ekraną
mov ax, 0002h
int 10h
;Išvedame praneėšimus ekrane
mov ah,09h
lea dx,aut      ;pirmo pranešimo adresas
int 21h
mov ah, 09h
lea dx, abc     ;antro pranešimo adresas
int 21h
mov ah, 09h
```

```
lea dx, txt       ;trečio pranešimo adresas
int 21h
;Atliekame veiksmus
SUB ax,ax      ;išvalome ax registrą
;pasirenkame programos kelia
CMP   a,ten
JG      daugiau
mov al,x        ;al:=a   , nusiunčiame x reikšmę į registrą
sub al,y        ; al:=x-y atimame skaičius
JMP Pasruos
Daudiau:
Mov AL,2
MUL   a         ;rezultatas yra AX
SUB BX,BX
MOV   BL,x
ADD   BL,y    ;rezultatas yra x+y
MUL   BL       ;rezultats yra AX

Paruos:
lea si, ats+4     ;patalpiname atsakymo adresą
MOV   CX,5
ciklas:           ;paruošiame rezultatą išvedimui, suformuodami ASCI kode
div ten           ;daliname iš dešimt
add ah, 30h     ;
mov [si],ah     ;masyvo skaitliukas
dec si            ;mažinam skaitliukąa vienetu
mov ah, 0       ;išvalom ah:=0
loop ciklas     ;grįžtame į ciklo pradžią
CMP   a,ten
JG      F1sp
MOV   ah,09h
LEA    DX,F2
JMP    Atsk
F1sp:
LEA    DX,F1
Atsk:
INT     21h
mov ah, 09h
lea dx, ats       ;atspausdina atsakymą
int 21h
;-grązžna į DOS
mov ah,4ch
int 21h
program ends
end start
```

II sutransliuojame programą

III Sulinkuojame programą