Kompiuterių Architektūros konspektas

Benediktas G. VU MIF, 2011-2015m
(radus netikslumų, turint klausimų rašyti** benediktog@gmail.com**)

Šios versijos data yra: *2017-10-04*
Naujausią šio konspekto versiją galima rasti adresu:
https://docs.google.com/document/d/1UzYBaxLhF8pqvRaBMstj6RDrV5SsCjkfNLw0f4K-r7Y/edit

Turinys

Kas yra šis konspektas?

Tai yra esminių kompiuterių architektūros dalyko niuansų santrauka. Šis konspektas teoriškai padengia visą kursą – įsisavinę jame pateiktą turinį turėtume gerai ar net puikiai išlaikyti egzaminą.

Vis tik pilnam dalyko vaizdui susidaryti stipriai rekomenduoju naudotis ir dėstytojų pateikiama medžiaga. *Šis konspektas yra tik jau ir taip gaunamos paskaitų informacijos* ***papildinys, ne pakaitalas*****.**

Šis konspektas yra mano studentiška iniciatyva, kelionei po komparch'o labirintus palengvinti, bet juos praeiti vis tiek reikės patiems.

Smagių akimirkų tyrinėjant šį vieną smagiausių (nors ir sudėtingiausių) dalykų! :), prireikus galite drąsiai kreiptis *benediktog@gmail.com* – vis tik jokių garantijų dėl to, kada ir ar išvis atrašysiu, bet labiausiai tikėtina, kad atsakymą gausite per dieną, dvi, o daugų daugiausiai – savaitę.

Geriausi linkėjimai,
Benediktas G.

Kaip jį skaityti?

Geriausia – nuosekliai. Temų aprašymai sudėti taip, kad tai ką skaitysite suprasti užtektų, to ką jau skaitėte iki tol. Pradedama išvis – nuo paties paprasčiausio dalyko – skaičiavimo sistemų. Kadangi daugiausia susidursite su dvejetaine ir šešioliktaine skaičiavimo sistemomis, nors šiaip jau kasdieniame gyvenime esame pratę prie dešimtainės – išmoksite skaičius rašyti ir tomis kitomis.

Tuomet, kai jau turėsite suvokimo, kas yra procesorius ir kas yra atmintis, kaip jie maždaug veikia, galėsite praktiškai visą likusį laiką skirti gilinti suvokimui, kaip iš tikrųjų veikia procesorius ir atmintis, kokios jų galimybės ir pan.

Ar man reikalinga „Kompiuterių architektūra“ gyvenime?

Ir taip, ir ne.

Priklausomai nuo to, ką veiksite – jei dirbsite darbą susijusį su kompiuteriais – pravers.

Kiek? → Programuotojams pravers labiausiai (lyginant su kitais darbais pvz. poreikių analizė, testavimas, vadovavimas IT projektams ir t.t.).

Kuo? → Visos programos, kurias kada nors rašysite bus tam tikrame lygmenyje verčiamos į asemblerį ir vykdomos procesoriaus. C, C++ ir pan. kalbos yra kompiliuojamos vadinasi, tai ką jomis parašysite iškart pavirs vykdomaisiais failais („Windows“ aplinkoje – exe, „Linux“ – elf) ir bus vykdoma procesoriaus. Prieš paverčiant į vykdomąjį failą kompiliatorius susigeneruos asemblerio variantą ir šį sukompiliuos (to net nepastebėsite, bet tai vyks).

Kitos kalbos – interpretuojamos – bet patys interpretatoriai buvo sukompiliuoti ir jie tėra programos, kurios analizuoja informaciją iš failo ir vyksta kaip įprastos sukompiliuotos programos, tik interpretavimo procesas yra labai neefektyvus.

Taigi visos problemos, kurias spręsite visada remsis į asemblerį, nors tiesiogiai su juo dirbti neteks, bet suvokimas apie asemblerį duos stiprų pranašumą prieš tuos, kurie prie aukšto lygio (artimų žmogui) programavimo kalbų prisėdo neturėdami asemblerio suvokimo.

Tai yra pagrindas visam tam, ką išmoksite programavime vėliau, nes vienu ar kitu metu visas kodas turi tapti mašininiu kodu, kad jį mašina (procesorius) galėtų vykdyti.

Dėstytojai yra išsakę keletą pagrindinių dalyko tikslų, kuriuos verta paminėti ir man:

Studentai turi suprasti, kad:

- Kompiuteris yra tik mašina gebanti vykdyti mašinines komandas ir pati nemąsto.
- Svarbiau šiuolaikinių technologijų pasaulyje yra mokėti suprasti problemą ir surasti būdus jai išspręsti, bei juos pritaikyti. Yra žmonės, kurie sugeba taikyti žinias ir yra žmonės, kurie geba kaupti žinias – bet tik maža dalis sugeba apjungti ir tai, ir tai – tačiau tai daugiau įdirbio, nei talento klausimas.
- Šis dalykas yra sudėtingas tiek, kad būtų iššūkis, bet įveikiamas iššūkis. Skyrę dėmesio – jį išmoksite ne tik kaip kažkokį dalyką, bet ir pagerinsite savo įgūdžius, kaip suprasti sudėtingus dalykus apskritai. Kiti sudėtingi dalykai po kompiuterių arcitektūros nebeturėtų gąsdinti. :)

Darbas su skaičiavimo sistemomis

Vertimas tarp skaičiavimo sistemų.

Pozicinėse skaičiavimo sistemose skaitmens (vieno skaitinę vertę turinčio simbolio) vertė tiesiogiai priklauso nuo jo pozicijos skaičiuje, pvz.: 123 ir 321 yra skirtingi skaičiai.

Žmonės įprastai naudoja dešimtainę pozicinę skaičiavimo sistemą.

Joje skaičius 123 atrodo taip:
$1*10^2+2*10^1+3*10^0=123$

Kartais dirbant su skirtingomis skaičiavimo sistemomis norint nesusipainioti su kuria sistema dirbama, po skaičiaus užrašomas indeksas, parodantis kokia tai sistema. Pavyzdžiui 123_{10} reiškia skaičių 123 dešimtainėje skaičiavimo sistemoje, 456_9 reiškia skaičių 456 devintainėje skaičiavimo sistemoje ir pan.

Programuojant asembleriu dažniausiai tenka susidurti su šešioliktaine, dvejetaine ir dešimtaine skaičiavimo sistemomis (kartais ir aštuntaine), indeksus rašyti programiniam kode arba neįmanoma, arba pernelyg sudėtinga, todėl po skaitmens tiesiog parašoma raidė, iš kurios transliatorius (paprastas kompiliatorius atliekantis nesudėtingas operacijas, iš esmės tik paverčiantis asemblerinį kodą tiesiogiai į jo mašininio kodo užrašymą) nusprendžia, kokioje skaičiavimo sistemoje užrašyti duotąjį skaičių.

Jokios raidės, pvz skaičius 19 reiškia dešimtainį skaičių 19

Po skaičiaus einanti ***h raidė, reiškia, kad skaičius pateiktas šešioliktainėje sistemoje***. 19h reiškia, kad tai yra šešioliktainis skaičius 19. Dešimtainėje sistemoje to paties skaičiaus reikšmė yra 25. Kaip verstis tarp sistemų aprašyta tolesniuose skyreliuose.

Po skaičiaus einanti ***b raidė, reiškia, kad skaičius pateiktas dvejetainėje sistemoje***. 101b tai yra dvejetainis skaičius 101. Jo reikšmė dešimtainėje sistemoje yra 5.

Vis tik svarbu žinoti ir tai, kad egzaminuose registrų ar baitų reikšmės paprastai pateikiamos baituose ar žodžiuose be h raidės ir nenurodant, kokia tai skaičiavimo sistema. Tada tiesiog juos reikia interpretuoti, kaip pateiktus šešioliktainėje sistemoje, jei nenurodyta kitaip.

Vertimas iš n-tainės į dešimtainę

Galioja tas pats, kas praeitame pavyzdyje su skaičiumi 123, tik vietoj 10 reikia panaudoti skaičiukus n.

Ką daryti su skaičiaus n laipsniais laipsniais?

Galima įsivaizduoti, kad po 123 yra kablelis, pavyzdžiui, tai skaičius 123,0000 (šiuo atveju nuliai po kablelio nieko nereiškia)

Tada reikia žinoti, kad skaitmenims einantiems nuo kablelio į kairę pusę reikia laipsnio rodiklį didinti po 1 pradedant nuliumi. Skaitmenims dešinėje kablelio pusėje į dešinę pusę laipsnio rodiklį mažinti pradedant nuo -1. Toliau pateikiamas to pavyzdys:

Tarkim turime skaičių ***241,64*** septintainėje pozicinėje (skaitmenys tik nuo 0 iki 6, nes septintainė sistema, kaip kad dešimtainėje nuo 0 iki 9) skaičiavimo sistemoje (n=7).

Pozicija nurodo kokiu laipsniu pakelti skaičiuką n.

n pakeltas pozicijos laipsniu padauginamas iš skaitmens esančio toje pozicijoje reikšmės. Gauti rezultatai sudedami, taip gauname tą patį skaičių tik jau dešimtainėje sistemoje.

Skaitmenų reikšmės	*2*	*4*	*1*	***, (kablelis)***	*6*	*4*
Pozicijos	***2***	***1***	***0***	***<<didėja nuo 0 >>mažėja nuo -1***	***-1***	***-2***

Dešimtainėje tai bus:

$2*7^2 + 4*7^1 + 1*7^0 + 6*7^{-1} + 4*7^{-2}=$
$= 2*49 + 4*7 + 1*1 + 6/7 + 4/49 =$
$= 98 + 28 + 1 + 6/7 + 4/49 =$
$127 + 42/49 + 4/49 = 127 + 46/49$

Vertimas iš dešimtainės į n-tainę

Bendruoju atveju algoritmas toks: dešimtainį skaičių daliname iš n per eilę žingsnių, kitame jau dalindami prieš tai atlikto dalybos veiksmo rezultatą, tiek kartų, kol dalybos rezultatas pasidaro lygus nuliui. Tada liekanas surašome atvirkščia tvarka negu gavome, gautas skaičius ir bus tas pats skaičius kitoje skaičiavimo sistemoje.

Pavyzdžiui, norime dešimtainį skaičių 502 paversti į n-tainę skaičiavimo sistemą.
Tarkim n=7 (t.y. versime į septintainę skaičiavimo sistemą)

502 / 7 = 71, liekana **5**
71 / 7 = 10, liekana **1**
10 / 7 = 1, liekana **3**
1 / 7 = ***0***, liekana **1** (***rezultatas nulis***, veiksmai toliau nebeatliekami)

Gautas skaičius septintainėj sistemoj yra 1315
Tai galime užrašyti tiesiog taip $502_{10}=1315_{7}$

Su sveikais skaičiais veiksmai atliekami taip kaip parodyta pavyzdyje prieš tai. Vis dėl to jei turime realųjį skaičių, tai su jo sveikąja dalimi atliekame prieš tai aprašytus veiksmus, o su dalimi po kablelio (vadinama trupmenine dalimi) dirbame atskirai.
Pavyzdžiui, verčiame dešimtainį skaičių $502,2356_{10}$ į septintainę pozicinę skaičiavimo sistemą.

Reikia dirbti su sveikąja ir trupmenine dalimi atskirai.
Su sveikąja dalimi jau išsprendėmė ir turime rezultatą 1315_{7}

Atmetę sveikąją dalį turime skaičiaus trupmeninę dalį 0,2356
Versdami į n-tainę:

- Sudauginame prieš tai buvusį rezultatą iš n
- Užrašome gauto skaičiaus sveikąją dalį
- Atmetę gautojo skaičiaus sveikąją dalį dauginame iš n vėl reikiamą kiekį kartų:
  - kol pastebėsime ciklišKumą – tada galėsime užrašyti periodą (į skliaustelius)
  - tiek kiek reikia uždaviniui išspręsti, mantisei užrašyti ir pan.

Tarkim šiuo atveju mus domina tik 5 skaitmenys po kablelio:
0,2356 * 7 = ***1***,6492
0,6492 * 7 = ***4***,5444
0,5444 * 7 = ***3***,8108
0,8108 * 7 = ***5***,6756
0,6756 * 7 = ***4***,7292

Tai atsakymas bus, kad $502,2356_{10}$ yra apytiksliai (ne tiksliai, nes rašėme tik 5 skaitmenis po kablelio, ir neatlikome apvalinimo) $1315,{\boldsymbol{14354}}_{7}$
Noriu pabrėžti, kad dirbant su trupmenine dalimi skaitmenis surašome ta pačia eilės tvarka, kuria gavome (kitaip nei dirbdami su sveikąja skaičiaus dalimi, kur liekanas rašėme atvirkščia tvarka, nei jos buvo gautos)

Pastabos:

- n (jei skaičius n parodo kelintainė sistema) visada yra natūralus skaičius didesnis už vieną
- keičiant neigiamo skaičiaus skaičiavimo sistemą, dirbama kaip su teigiamu, bet gavus rezultatą minusas vėl uždedamas. Vis dėl to, dirbant su ribotais atminties laukais **(baitais, žodžiais)** neigiamų skaičių nėra, yra tik vienetų ir nulių sekos, todėl tada **dirbant su neigiamais skaičiais galioja kitos taisyklės**. Plačiau tai išnagrinėta skaičių su ženklu ir be ženklo temoje.

Vertimas tarp „bendro pagrindo“ skaičiavimo sistemų

Verčiant tarp skaičiavimo sistemų dažniausiai tenka panaudoti dešimtainę kaip tarpinę (t.y. iš 7tainės versdami į 9tainę pirmiau 7tainį skaičių pasiverčiame 10tainiu, o tik tada gautą 10tainį verčiame 9tainiu)

Tačiau taip daryti ne visada reikalinga:
Sakykime turime dvi sistemas, p ir q
Taip pat pasižymime p ir q taip, kad p>q (turėdami priešingai galime skaičius sukeisti)

Kai $p=a^k$ ir $q=a^m$ turi galioti sąlygos:

- a, k, m – natūralieji skaičiai, a>1
- k dalybos iš m liekana yra nulis (t.y. k mod m=0)

Tada skaičiavimo sistemas (šiame konspekte) vadinkime tiesiog „bendro pagrindo“ arba „suderintomis“

Jose kiekvieną p sistemoje užrašyto skaičiaus skaitmenį galima išskleisti į (k div m) skaitmenų q sistemoje ir atvirkščiai (reikia pradėti tai daryti nuo dešinės pusės, jei kairėje pritrūktų skaitmenų galima prisirašyti nulius).

Nagrinėjant kompiuterių architektūrą paprastai dirbama su sistemomis, kur a=2.
Pavyzdžiui: bendro pagrindo sistemos yra 16-tainė ir 2-tainė
Šiuo atveju:
a=2, k=4, m=1

k div m=4 div 1=**4**, (4 mod 1=0) t.y. dalybos 4/1 liekana yra 0.
Šiuo atveju kiekvieną skaitmenį 16-tainėje sistemoje galima išskleisti į **4** dvejetainius skaitmenis ir atvirkščiai.

2				***E***			
0	***0***	***1***	***0***	***1***	***1***	***1***	***0***

Šešioliktainis skaičius 2E atrodytų taip (žr. lentelę aukščiau), taip pat šį skaičių galima gauti iš dvejetainės sistemos (t.y. galimi veiksmai ir į vieną ir į kitą pusę: iš šešioliktainės į dvejetainę ir atvirkščiai).

- Viršutinėje eilutėje yra šešioliktainis skaičius
- Apatinėje eilutėje yra dvejetainis skaičius

10tainė	***2tainė***	***16tainė***
0	**00**00	0
1	0001	1
2	0010	2
3	0011	3
4	**01**00	4
5	0101	5
6	0110	6
7	0111	7

10tainė	***2tainė***	***16tainė***
8	**10**00	8
9	1001	9
10	1010	A
11	1011	B
12	**11**00	C
13	1101	D
14	1110	E
15	1111	F

Nesunku pastebėti, kad einant nuo dešinės pusės į kairę dvejetainių skaitmenų svoriai yra tokie: 1, 2, 4, 8 ir t.t. (didėja po 2 kartus, sistema 2-jetainė, 0*svoris=0 ir 1*svoris=svoris)
T.y. jei 4 dvejetainiai skaitmenys yra wxyz tai dešimtainė reikšmė gaunama: w*8 + x*4 + y*2 + z

Dvejetainiai skaičiai

Apie dvejetainius skaičius ir dvejetainių vertimas dešimtainiais

Paprastai veiksmai atliekami su baitais arba žodžiais.

Bitas – vienas dvejetainis skaitmuo (0 arba 1), mažiausia nedaloma atminties dalis.

Baitas – 8 bitų rinkinys (0..255 skaičius be ženklo) arba (-128..+127 skaičius su ženklu)

Žodis – 16 bitų rinkinys (0..65535 skaičius be ženklo) arba (-32768..+32767 skaičius su ženklu)

Rezultato laukas – tam tikrų aritmetinių/loginių veiksmų vykdymo metu gauto rezultato jauniausi 8 arba 16 bitų, priklausomai nuo to operacija atlikta su baitais ar žodžiais (Carry Flag bitas rezultato laukui nepriklauso).

Šiame skyrelyje pavyzdžiai pateikiami su baitais, bet žodžiams galioja tos pačios taisyklės.

Bendruoju atveju:

Kai lauką sudaro n ***bitų***: (vieno baito atveju n=8)

Skirtingų reikšmių gali būti 2^n (vieno baito atveju 256)

Skaičiuose be ženklo reikšmių diapazonas $0..2^n-1$ (vieno baito atveju 0..255)

Skaičiuosi su ženklu reikšmių diapazonas $-2^{n-1}..+2^{n-1}-1$ (vieno baito atveju -128..127)

Kaip nustatyti, kokia skaičiaus reikšmė (be ženklo) yra dešimtainėje sistemoje turint dvejetainį užrašą?

Pavyzdys:

Turime lentelę, kuria apsirašome baitą, šioje lentelėje:

- Pirma eilutė reiškia bitų reikšmes
- Antra eilutė – skaitmens poziciją (iš dešinės į kairę nuo 0 iki n-1, 8bitai tai nuo 0 iki 7tos)
- Trečia skaitmens svorį (2 pakelta pozicijos laipsniu)

0	1	1	0	1	1	0	1
7	***6***	***5***	***4***	***3***	***2***	***1***	***0***
128	64	32	16	8	4	2	1

Norėdami nustatyti kokia čia dešimtainė reikšmė užkoduota (skaičius BE ŽENKLO!) sudedame svorius bitukų, kurių reikšmės yra vienetai (nulius sudėti ne taip jau ir prasminga, vis tiek rezultatas nesikeis).

Turime: $64+32+8+4+1=109_{10}$.

Tie patys veiksmai detaliau (visi skaičiavimai dešimtainėje sistemoje):

$0*2^7 + 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 =$

$= 0*128 + 1*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 1*1 =$

$= 0 + 64 + 32 + 0 + 8 + 4 + 0 + 1 =$

$= 64 + 32 + 8 + 4 + 1 = 109_{10}$

Turėdami skaičiaus be ženklo reikšmę, nesunkiai galima pasiversti jį skaičiumi su ženklu ir atvirkščiai.

Dešimtainių vertimas dvejetainiais

Pavyzdžiui, turime dešimtainį skaičių. Norime jį užkoduoti į kažkiek baitų dvejetainiu kodu. Jei turime neigiamą skaičių, jį pirmiausia pasiverčiame skaičiumi be ženklo.

Principas:

Einame nuo didžiausios pozicijos link nulinės ir lyginame pozicijos svorį, su turima reikšme.

- Jei turima reikšmė didesnė arba lygi tos pozicijos svoriui, toje pozicijoje, tai iš turimos reikšmės atimame skaitmens svorį, toje vietoje įrašome 1tuką ir einame į dešinę.
- Jei turima reikšmė mažesnė už to skaitmens svorį, toje pozicijoje rašome **0uką** ir einame į dešinę **nekeisdami turimos reikšmės.**
- Kai turima reikšmė yra 0 baigiame darbą, jei praėję iki nulinės pozicijos turime ne nulinį skaičių reiškia kažkur padarėme klaidą arba duotas dešimtainis skaičius netelpa tokio dydžio atminties lauke (vienam baite maksimali reikšmė 255, dviejuose baituose 65535).

Pavyzdys:

Turime dešimtainį skaičių be ženklo 159. Norime jį užrašyti viename baite dvejetaine skaičiavimo sistema.

- Lyginame 159 ir 128. 159>128 todėl rašome **1**.
  - Turima reikšmė 159-128=31
- Lyginame 31 ir 64. 31<64, todėl rašome **0**.
  - Turima reikšmė išlieka 31
- Lyginame 31 ir 32, 31<32, todėl rašome **0**
  - Turima reikšmė išlieka 31
- Lyginame 31 ir 16, 31>16, todėl rašome **1**
  - Turima reikšmė 31-16=15
- Lyginame 15 ir 8, 15>8, todėl rašome **1**
  - Turima reikšmė 15-8=7
- Lyginame 7 ir 4, 7>4, todėl rašome **1**
  - Turima reikšmė 7-4=3
- Lyginame 3 ir 2, 3>2 todėl rašome **1**
  - Turima reikšmė 3-2=1
- Lyginame 1 ir 1, 1=1, todėl rašome **1**
  - Turima reikšmė 0, pasiekta nulinė pozicija, klaidų nėra.

Vadinasi: $159_{10} = 1001\ 1111_2$

Kitas būdas:

Galima skaičių BE ženklo, 159 versti į dvejetainę ir kitu būdu:
Jei turėtume žodį, turėtume dalinti iš 256, ir gautume atskirų baitų reikšmes (rezultatas vyresnysis baitas, liekana jaunesnysis).
Turėdami baitą jį galima užrašyti dviem šešioliktainiais (skaldymas į pusbaičius).
159/16 = ***9***, liekana ***15***. 15 šešioliktainėje reiškią raidę F. Todėl tai yra šešioliktainis skaičius ***9F***
Šešioliktainį išsiskleidžiame į dvejetainį: 9h=1001, Fh=1111

Vadinasi: $159_{10} = 1001\ 1111_2$

*Turint dešimtainius **neigiamus** ir norint užrašyti dvejetainėje sistemoje* (iš to lengvai galima pasiversti ir į šešioliktainę ar aštuntainę) *viename baite arba žodyje galimi tokie to atlikimo būdai*:

Pirmas būdas:

Tą patį dešimtainį skaičių pasiverčiame skaičiumi be ženklo („pritempiame iki skaičiaus be ženklo intervalo“) ir paverčiame į dvejetainę sistemą vienu iš prieš tai nurodytų būdų.

Antras būdas:

Ignoruojame turimą minusą prieš skaičių, turimą skaičių užrašome dvejetaine sistema (pagal prieš tai nurodytus veiksmus) ir pakeičiame jo ženklą (t.y. visus baito arba žodžio bitus invertuojame ir pridedame vienetą).

Skaičiai su ženklu ir be ženklo

Bet kurio rezultato lauko (baito, žodžio) reikšmę galima interpretuoti ir kaip skaičių su ženklu, ir kaip skaičių be ženklo.

Skaičiaus su ženklu ženklą parodo vyriausias (pats pirmas kairėje) baito arba žodžio *bitas*.

- Kai jo reikšmė 0, ženklas +
- Kai jo reikšmė 1, ženklas -

Viename baite (tie patys principai galioja ir užrašant per daugiau baitų, žodžiuose, dvigubuose žodžiuose ir pan.):
0 yra ***0000 0000***
1 yra ***0000 0001***
-***1*** yra ***1111 1111***
Taip yra todėl, kad sudėjus du skaičius su ženklu turime gauti teisingą rezultatą
Viename baite galima išsaugoti vieną iš 256 skirtingų reikšmių: 128 neigiamus, ir 128 teigiamus skaičius intervalas yra [-128; 127], kur nulio ženklas yra +.
Pavyzdžiui, dešimtainis 130 be ženklo dvejetainėj yra 1000 0010, su ženklu tai būtų -126.

Pasivertimas į skaičių BE ženklo arba SU ženklu (reikšmė dvejetainiam pavidale nesikeičia)

Turime sveikąjį skaičių x.

Žinome jo reikšmę, ir norime užrašyti jį viename baite dešimtaine sistema (iš dešimtainės paprasta gauti ir kitas).

Skyrelio pradžioje paminėti intervalai:

Viename baite sveikas skaičius be ženklo priklauso intervalui ***[0; 255]***

Viename baite sveikas skaičius su ženklu priklauso intervalui ***[-128; 127]***

Reiškia reikia privesti skaičių x iki kažkurio iš šių intervalų, priklausomai nuo to ar reikalingas skaičius su ženklu ar be ženklo.

Kadangi n bitų laukai visi yra riboti (baito atveju ***n=8***), tai galima pridėti/atimti iš skaičiaus x, skaičių 2^n tiek kartų, kiek reikia, kol šis pakliūva į norimą intervalą (skaičiaus su ženklu arba be ženklo).

To paaiškinimas būtų toks:

Skaičių ribotame atminties lauke galima vienu metu laikyti ir skaičiumi su ženklu, ir skaičiumi be ženklo, jei pridėtume arba atimtume po vienetą 2^n kartų vėl gautume tą patį rezultatą ribotame atminties lauke, nes „tai kas išlenda už rezultato lauko ribų, perteklius“ įtakos rezultatui ribotame lauke neturi, taip pat turėdami visą rezultato lauką užpilytą vienetiniais bitais ir pridėję vieną gautume nulį ir atvirkščiai. Taip pat galioja iš pažiūros neįprastos savybės, kad viename baite FF+01=00 ir 00-01=FF

Iš tikrųjų pridėdami arba atimdami po vienetą 2^n kartų mes tarpiniuose skaičiavimuose gauname visas įmanomas reikšmes tame apribotame rezultato lauke „apsukame ratą“ ir vėl grįžtame prie pradinės, tai faktiškai nieko ir nepakeičiame, tik dešimtainiam pavidale jau turime patogesnę reikšmę, be to priklausančią norimam – skaičiaus su ženklu ar be ženklo intervalui.

Pavyzdžiui, skaičius x yra -189 dešimtainėje, viename baite (bitų kiekis n=8, galimų reikšmių 2^8=***256***) skaičius be ženklo bus -189+***256***=67. Jis patenka ir į skaičiaus su ženklo intervalą, todėl ir skaičius su ženklo ir be ženklo dešimtainėje sistemoje bus 67. šešioliktainėje šis skaičius yra 43, dvejetainėje 0100 0011.

Ženklo keitimas (tik skaičiams SU ženklu) – dvejetainiam pavidale gauname priešingo ženklo reikšmę

Turint *dvejetainę* išraišką atliekame du veiksmus:

- Invertuojame visus bitus (nulius pakeisti vienetais, vienetus nuliais)
- Pridedame vienetą

Svarbu:

- Sprendžiant įvairius uždavinius reikia žinoti kada skaičių reikia interpretuoti kaip skaičių su ženklu, o kada kaip skaičių be ženklo.

Pvz.: Tikrinant ar aritmetikoje įvyko overflow turime žiūrėti į skaičius ir rezultatą kaip skaičius su ženklu, gavus klaidingą rezultatą ***OF=1***

- 1baito skaičiai dažniausiai yra suprantami kaip skaičiai su ženklu, norint užrašyti tą patį skaičių dviejuose baituose reikia praplėsti pagal ženklo bitą. (vyresniame baite sudėti 1tukus arba 0, tokius koks yra ženklo bitas). To dažnai prireikia skaičiuojant poslinkį kodo segmente, kai jis yra vieno baito, o IP registras dviejų baitų.

Baitų (arba žodžių) sudėties ir atimties operacijos

Su baitais, žodžiais (2baitai), dvigubais žodžiais (4baitai) ir t.t. galima atlikti įvairius loginius ir aritmetinius veiksmus.

Šiame skyrelyje aprašytos dvejetainės ir šešioliktainės baitų sudėties ir atimties aritmetinės operacijos.

Sudedant/atimant skirtingo dydžio laukus reikia mažesnįjį privesti prie didesniojo, t.y, suvienodinti jų dydžius, skaičiuojant poslinkius tam naudojama ženklo plėtimo taisyklė aprašyta šiame skyrelyje ***(čia)***.

Baitų sudėtis dvejetainėje sistemoje

Sudėtis vykdoma iš dešinės į kairę, sudedamos bitų poros. Jei reikia atliekamas pernešimas (priešingas veiksmas pasiskolinimui atimtyje).

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0, pernešimas į sekantį (kairiau esantį) bitą, jei to bito nėra galima iš kairės prisirašyti nulinių bitų kiek tik reikia, jei kam nuo to aiškiau, nes tai rezultato nekeičia.

Išnagrinėkime **pavyzdį**:
1 baito operacijomis sudedami 156 ir -23.

1. Pirmiausia užrašome šiuos skaičius dvejetainėje sistemoje:

   Kadangi 156 mažiau už 255 reiškia telpa vienam baite, skaičius teigiamas ir patenka į skaičiaus be ženklo intervalą, todėl elgsimės kaip su skaičiu be ženklo (išsirašome per dvejeto laipsnių sumą, kad žinotume kur sudėti nulius, o kur vienetus 156=128+16+8+4)

   Kadangi -23 daugiau už -128 reiškia telpa vienam baite, skaičius neigiamas todėl pirmiausia pasiverčiame jį į skaičių be ženklo. Dirbama su 1 baitu, t.y. 8 bitai, o 2^8=256. Pridėję 256 prie -23 gauname 233 ir tai jau yra teisingas (toks pat) skaičius BE ženklo.
   233=128+64+32+8+1

Atliekame sudėties operaciją su šiais skaičiais:
Eilutėse:

1. Surašytos dešimtainio skaičiaus 156 dvejetainio užrašo bitų reikšmės
2. Surašytos dešimtainio skaičiaus -23 bitų dvejetainio užrašo reikšmės
3. Surašytos rezultato bitukų reikšmės (kairė pozicija papildomas bitas, kuris dedamas į CF bitą Status Flag registre, žalias langelis)

	1	0	0	1	1	1	0	0
+	1	1	1	0	1	0	0	1
1	1	0	0	0	0	1	0	1

Atliekame veiksmus nuo dešinės į kairę.
0 + 1 = 1
0 + 0 = 0
1 + 0 = 1
1 + 1 = 0 (pernešimas į sekančią bitų pora, ji padidinama vienetu), dėl pernešimo būtų fiksuojamas flagas AF=1

1 + 0 + 1 (perneštas bitas) = 0, suma didesnė už 1, vėl įvyko pernešimas
0 + 1 + 1 (perneštas bitas) = 0, suma didesnė už 1, dar vienas pernešimas
0 + 1 + 1 (perneštas bitas) = 0
1 + 1 +1 (perneštas bitas) =1, suma didesnė už 1, įvyko pernešimas į papildomą bitą (Carry Flag'ą)

Baitų atimtis dvejetainėje sistemoje

Atimtis, kaip ir sudėtis vykdoma bitas po bito iš dešinės į kairę, esant reikalui (atimant didesnį iš mažesnio)

0 – 0 = 0
1 – 0 = 1
1 – 1 = 0
0 – 1 = 1 pasiskolinimas (vykdomas skolinimas iš kairiau esančios bitų poros, jei jos nėra, galima įsivaizduoti, kad ta bitų pora yra 0 ir 0)

Išnagrinėkime pavyzdį:

Vieno baito atimties operaciją atlikime su skaičiai 25 ir 142.
Pirmiausia pasiverčiame šiuos skaičius į dvejetainę sistemą ir užrašome viename baite:
25_{10}=0001 1001_2
142_{10}=1000 1110_2

Atliekame atimties operaciją su šiais skaičiais:
Eilutėse:

1. Surašytos 25 bitų reikšmės
2. Surašytos 142 bitų reikšmės
3. Surašytos rezultato bitų reikšmės (kairė pozicija papildomas bitas, kuris dedamas į CF bitą Status Flag (SF) registre, žalias langelis)

___	0	0	0	1	1	0	0	1
	1	0	0	0	1	1	1	0
1	1	0	0	0	1	0	1	1

Atliekame veiksmus nuo dešinės į kairę.
Jei yra vykdomas pasiskolinimas, skolos skaičiukas pateiktas raudonu šriftu.
1 – 0 = 1
0 – 1 = 1 (iš mažesnio atimamas didesnis, todėl skolinamės iš kairiau esančios skaičių poros)
0 – 1 – 1 = 0 (iš mažesnio atimamas didesnis, todėl skolinamės iš kairiau esančios skaičių poros)
1 – 1 – 1 = 1 (iš mažesnio atimamas didesnis, todėl skolinamės iš kairiau esančios skaičių poros **1-2**)

1 – 0 – 1 = 0
0 – 0 = 0
0 – 0 = 0
0 – 1 = 1
Taip pat, kadangi iš mažesnio skaičiaus (laikant, kad abu be ženklo) atimamas didesnis rezultatas nebetilps skaičiaus be ženklo intervale [0;255], būtent dėl tos priežasties atliekant atimtį aštuntoje iš dešinės poroje prireikia pasiskolinimo „iš už rezultato lauko ribų“, todėl flagas CF=1.

Paprastai atimtis ir skolinimaisi yra sunkiai perprantami, ir jei kam nors atrodytų aiškiau tai galima iliustruoti, tokiu pavyzdžiu:
Skolos perduodamos iš kartos į kartą. Žmogus A pasiskolino, bet negrąžino skolos, todėl paveldėjo jo vaikas B, šis neapmokėjo, tai perėjo B vaikui C ir tik vaikas D apmokėjo skolą, todėl D vaikui E šia skola rūpintis nebeteks, nebent jis pats skolinsis ir perduos tą „naštą“ savo vaikams. Panašiai ir dvejetainiuose skaičiuose, kur galima įsivaizduoti, kad kairiau esantis bitas yra konkretaus bito vaikas ir t.t.

Žodžių sudėtis ir atimtis šešioliktainėje sistemoje

Veiksmai atliekami atsižvelgiant į tas pačias taisykles, kaip ir sudėties/atimties dešimtainėje sistemoje atveju (vienas iš patogiausių būdų – sudėtis stulpeliu).

Vienintelis pastebimas skirtumas, kad yra ne tik skaitmenys nuo 0 iki 9, bet ir raidės A..F reiškia skaitmenis. Reikia žinoti jų vertes:

***A**=10*
***B**=11*
***C**=12*
***D**=13*
***E**=14*
***F**=15*

Tuomet sudedant arba atimant, pernešimas/pasiskolinimas įvyksta ne peržengus dešimtį, 16 (sistema nebe dešimtainė, o šešioliktainė).
Pvz.: sudedam skaitmenis x ir y.
Rezultate rašom (x+y) mod 16
Kitoj poroj pridedam (x+y) div 16 („pernešimas į kitą porą, t.y. dviejų skaitmenų sudėties perteklių turėsime minty“)

SVARBU žinoti, kad efektyvus adresas visada užima 4 skaitmenis, o absoliutus 5 skaitmenis. Jei gaunama daugiau skaitmenų, į papildomus dėmesio nekreipiama, jei skaitmenų trūksta reikia papildyti nuliais priekyje.
Tačiau skaičiuojant registrų sumą ribojami tik atskirų registrų dydžiai, bet ne jų suma.
Jei AX=FFFF, BX=0002, tai sumą rašytume 10001, o ne 0001 kaip efektyvaus adreso atveju.

Pavyzdys su sudėtim (vykdoma iš dešinės į kairę):

+	F	3	2	7
	8	4	E	F
1	7	8	1	6

Pernešimo vienetai pažymėti raudonai.

7 + F = (7+15=22, dalybos iš 16 rez. ***1*** (pridėsim prie poros kairiau), liekana 6 rašom į rezultatą)
2 + E + ***1*** = (3 + E = 3 + 14, dalybos iš 16 rez. ***1*** (pridėsim prie poros kairiau), liekana 1 rašom į rezultatą)
3 + 4 + ***1*** = 8 (dalybos iš 16 rez. 0, nieko nepridedam prie poros kairiau, liekana 8 rašom į rezultatą)
F + 8 = (15+8=23, dalybos iš 16 rez. ***1*** (pridėsim prie poros kairiau), liekana 7 rašom į rezultatą)
Įrašomas ***papildomas vienetas***. Jei buvo skaičiuotas efektyvus adresas, į šį vienetą reikia nekreipti dėmėsio (toliau jo nebenaudoti).

Pavyzdys su atimtim (vykdoma iš dešinės į kairę):

```
  2  9  C  D
— 3  A  F  9
-------------
  E  E  D  4
```

Pasiskolinimo bitukai pažymėti raudonai

Veiksmai atliekami iš dešinės į kairę:
D – 9 = 13 – 9 = 4 (jokio skolinimosi iš poros kairiau)
C – F = 12 – 15 = -3, (neigiamas sk. todėl 16-3=13=D, ir pasiskolinimas iš poros kairiau)
9 – A – ***1*** = 8 – A = 8 – 10 = -2 (neigiamas sk, todėl 16-2=14=E, ir pasiskolinimas iš poros kairiau)
2 – 3 – ***1*** = 2 – 4 = -2 (neigiamas sk, todėl 16-2=14=E, ir pasiskolinimas iš poros kairiau, kadangi šios nėra tai iš papildomos „įsivaizduojamos“ poros, kuri žodžio dydžio lauke vis tiek nebus išsaugota, o rezultato mums vis tiek reikės neigiamo)

Papildoma pora:
0 – 0 – 1 = 0 – 1 = -1, t.y. 16_{10}-1=15_{10}=F_{16}
Kadangi rezultatas skaičiuotas 4 skaitmenyse (2 baitai), tai į papildomas poras dėmesys nekreipiamas.

Šiuo atveju gautas skaičius būtų teisingas skaičiuojant 2 baituose, tačiau atliekant veiksmus ne apribotuose laukuose (baitas 2 skaitmenys, žodis 4 skaitmenys) reiktų pastebėti, kad atimamas didesnis iš mažesnio, todėl atsakymas bus neigiamas ir tada žinant, kad atsakymas neigiamas atimti atvirkščiai 3AF9-29CD. Tuomet gautume -112C.

Galima spręsti ir taip, tuomet gavę neigiamą skaičių dar turėtume pridėti prie 10000h.
10000h + (-112Ch) = EED4h
Nulių po vieneto yra tiek, kiek skaitmenų (baitas 2 skaitmenys, žodis 4 skaitmenys)

Kad gavome teisingai galime įsitikinti sudėdami EED4+3AF9=129CD
Vis tiek į perteklinį vienetą dėmesio nekreipiame (jei rezultatą saugome viename žodyje, tai tik 4 šešioliktainiai skaitmenys, arba jei skaičiavome efektyvų adresą)

Keletas ypatingesnių baitų ir žodžių reikšmių

Šiaip dešimtainis skaičius	Dešimtainis baite su ženklu	Dešimtainis baite be ženklo	Šešioliktainis baite su ženklu	Šešioliktainis baite be ženklo	Dešimtainis žodyje su ženklu	Dešimtainis žodyje be ženklo	Šešioliktainis žodyje su ženklu	Šešioliktainis žodyje be ženklo
-1	*-1*		*FF*		*-1*		*FFFF*	
-2	*-2*		*FE*		*-1*		*FFFE*	
-127	*-127*	-	*81*	-	*-127*	-	*FF81*	-
-128	*-128*		*80*		*-128*		*FF80*	
0	*0*	*0*	*00*	*00*	*0*	*0*	*0000*	*0000*
1	*1*	*1*	*01*	*01*	*1*	*1*	*0001*	*0001*
254	-	*254*	-	*FE*	*254*	*254*	*00FE*	*00FF*
255		*255*		*FF*	*255*	*255*	*00FF*	*00FF*
32767	*tokia reikšmė baite netelpa*				*32767*	*32767*	*7FFF*	*7FFF*
-32768					*-32768*	-	*8000*	-
32768					-	*32768*	-	*8000*
65535						*65535*		*FFFF*

Brūkšniukai reiškia, kad reikšmė nepatenka į tokio skaičiaus intervalą.

Architektūros ypatybės

Svarbiausi kompiuterio komponentai

Jei Jūsų paklaustų, kas svarbiausia kompiuteriui, ką atsakytumėt? Galbūt pagal savo pomėgius minėtumėte ausines, pelę, klaviatūrą, monitorių...

Bet jei to paties paklaustume kitame kontekste – be ko kompiuterio nebūtų išvis? Tuomet pastebėtume, kad gali būti jis ir be ausinių ir be pelės, ir be klaviatūros... Net be monitoriaus. Jį atjungus jis toliau gali vykdyti savo skaičiavimus, kurių greičiausiai nematysime, nebent būsime paleidę programą, kuri spausdina rezultatus – tada tai ištrauktume iš spausdintuvo...

Ką bandau pasakyti?

Kad norint išgryninti, be ko kompiuteris nėra kompiuteris liekame su dviem komponentais – atmintimi ir procesoriumi. Toliau visas konspektas yra apie tai, kaip procesorius bendrauja su atmintimi (išskyrus MPL skyrelį).

Procesorius vykdo komandas, o atmintyje saugomi duomenys (pačios komandos, stekas, kiti duomenys). Procesorius turi savo duomenis (registrus), kurie apibūdina procesoriaus būseną – pvz. nuorodas, kur atmintyje padėti duomenys, o kur vykdomasis kodas (kur esanti komanda bus vykdoma po dabartinės), keletas skaičiavimams aktualių registrų.

Esminės architektūros detalės

Mūsų nagrinėjamos architektūros ypatybės būdingos ***Intel 8086*** ir ***Intel 8088*** procesoriams.

Iš esmės jų architektūros yra panašios, tuo labiau, naudojamas tas pats instruction set (asemblerinių komandų rinkinys) *x86-16bit*.

Kiekvienam kompiuteriui <u>būtiniausios dalys yra procesorius ir atmintis</u>. Kompiuteris savo darbo metu sugeba tik reaguoti į įvarių įrenginių signalus, vykdyti programos kodą ir keisti atminties būseną (atskirų registrų ir atminties baitų reikšmes).

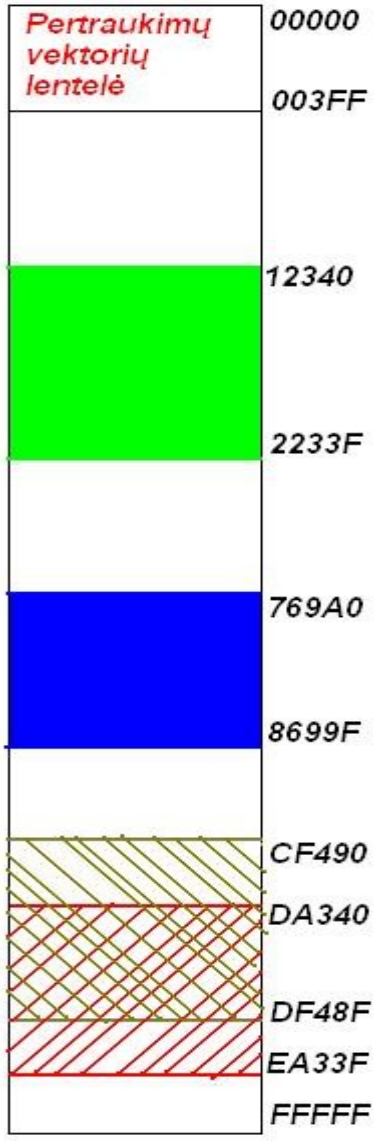
Mikroprocesoriuje išskiriamos keletas esminių dalių:

1. Registrai
2. ALU (Aritmetinis loginis įrenginys)
3. Vykdomojo ir absoliutaus adresų formavimo įrenginiai
4. Pertraukimų sistemos valdiklio (pertraukimų kontrolerio).

Dar viena kompiuterio architektūros dalį (ne paties mikroprocesoriaus dalį) išskirkime atmintį (RAM), kurioje saugomas tuo metu vykdomas kodas, duomenys, stekas ir pan.

Toliau detaliau nagrinėjama labiausiai programuotojui asembleriu aktuali kompiuterio dalis – procesoriaus registrai ir atmintis.

Atmintis ir jos segmentacija



Mūsų nagrinėjamoje architektūroje darbinė atmintis (kurioje) yra 1MB dydžio, t.y. $2^{20}=16^5$ baitų. Joje saugomi tuo metus vykdomos programos kodas, duomenys, stekas. Kiekvienas baitas atmintyje adresuojamas penkiais šešioliktainiais skaitmenimis, kaip, kad parodyta schemoje kairėje.

Pirmasis kilobaitas (1024 baitai) nuo 00000 iki 003FF yra užpildyti pertraukimų vektorių lentelės duomenimis. Tai yra CS ir IP registrų rinkiniai. Iškvietus tam tikrą pertraukimo procedūrą, jos adresas imamas būtent iš vektorių lentelės, ir vykdomas vektoriumi nurodytu adresu esantis kodas.

Tam kad būtų patogiau dirbti su atmintimi, ji yra skirstoma į segmentus. Šioje architektūroje paragrafu vadinsime 16 baitų bloką. Tuomet segmentinis registras rodo, į segmento pradžią, kuri yra adresu: segmento_registro_reikšmė * paragrafo dydis (t.y.10h)

Iš viso šioje architektūroje yra naudojami keturi segmentai:
CS – Code Segment – Segmentas, kuriame saugomas vykdomas mašininis kodas.
SS – Stack Segment – Steko segmentas, jame programa darbo metu „pasideda“ įvairias reikšmes.
DS, ES – Duomenų ir papildomas duomenų (extra) segmentai – saugomi duomenys, kuriuos apdoroja programa savo darbo metu.

Dalijimas segmentais realizuojamas segmentiniams registrams priskiriant 16bitų reikšmes. Pavyzdžiui, jei žaliame fone turime kodo segmentą, mėlyname duomenų, raudonom linijom steko, o rudom extrą segmentą, tai pagal paveikslėlį registrų reikšmės būtų:
CS=*1234*, ***DS***=*769A*, *ES*=*CF49*, ***SS***=*DA34*.

Segmentinį registrą padauginę iš 10h (paragrafo dydžio) gauname segmento pradžios *<u>absoliutų</u>* adresą. Pvz steko segmento DA340. Poslinkis segmente išreiškiamas *<u>efektyviu</u>* adresu nuo 0000 iki FFFF (kiekvieno dydis 10000h, arba tiesiog 64K).

Priklausomai nuo registrų reikšmių, gali ir persidengti (turėti bendrų laukų), ar net visiškai sutapti. Atmintyje tiek vykdomas kodas, tiek duomenys yra saugomi nulių ir vienetų sekomis ir nėra kažkaip ypatingai išskirti. Tiesiog įvairių registrų deriniais parodoma, kurias atminties vietas reikia vykdyti kaip kodą, kur duomenys ir pan.

Segmentai yra cikliniai: FFFF+1=0000, 0000-1=FFFF.
Atmintis irgi: FFFFF+1=00000, 00000-1=FFFFF.

Jei segmentas prasideda netoli atminties pabaigos, tai jo pabaigos dalis gali būti ir atminties pradžioje (dėl atminties ciklišumo, sekantis adresas po FFFFF yra vėl 00000)

Procesoriaus registrai

Registras – Vieno arba dviejų baitų laukas procesoriuje, veikiantis panašiai kaip atmintis, tik sparčiau. Registrai, kitaip nei atminties baitai/žodžiai nėra adresuojami.
Paragrafas – 16 baitų dydžio blokas atmintyje (kitose architektūrose šis dydis gali skirtis!)

Intel 8088 turi tokius registrus:

Darbiniai (skaičiavimam, reikšmėm pasidėti, dažnai naudojami kaip parametrai INT procedūroms):

- **AX** – vadinamas akumuliatorium (akumuliatorius vienam baite yra registras AL)
- **BX** – vadinamas baziniu registru (naudojamas adresavimui segmentuose)
- **CX** – vadinamas counteriu, ciklų skaitliuku ir pan.
- **DX** – vadinamas data registru

Segmentiniai (rodo kur atmintyje prasideda segmentas)***:***

- ***ES*** – rodo extra segmento pradžią (paragrafo atmintyje nr.) - *dar vadinamas papildomo duomenų segmento registru*
- ***CS*** – rodo kodo segmento pradžią (paragrafo atmintyje nr.)
- ***SS*** – rodo steko segmento pradžią (paragrafo atmintyje nr.)
- ***DS*** – rodo data segmento pradžią (paragrafo atmintyje nr.)

Paragrafas 1MB RAMuose (kaip kad Intel 8088 atveju) yra 16baitų dydžio blokas.

Indeksiniai (eilutinėms komandoms, duomenų persiuntimui, kopijavimui, adresavimui)

- ***SI*** – Source Index šaltinio indeksas dažniausiai naudojamas su DS
- ***DI*** – Destination Index – eilutinėse komandose tik su segmentu ES

Kiti:

- ***IP*** – Instruction Pointer, kartu CS rodo poslinkį nuo kodo segmento pradžios, ***komandos vykdymo metu rodo į sekančios vykdomos komandos pradžią.***
- ***BP*** – Base Pointer, kartu su SS naudojamas adresavimui steke.
- ***SP*** – Stack Pointer, kartu su SS rodo į steko viršūnę.
- ***SF*** – Status Flag, rodo paskutinės aritmetinės/loginės operacijos rezultato požymius (ženklą), naudojamas ir procesoriaus darbinio rėžimo reguliavimui. Vadinamas procesoriaus būsenos registru.

Darbiniuose registruose galima naudoti ir jų vyresniuosius arba jaunesnius baitus.
?X registro jaunesnysis baitas yra ?L (Low), ir vyresnis ?H (High)
Vietoj klaustuko {A, B, C, D}

Atliekant skaitymus iš atminties duomenys skaitomi pirmiau į jaunesnįjį baitą, tada į vyresnįjį!

AX	
AH	*AL*

BX	
BH	*BL*

CX	
CH	*CL*

DX	
DH	*DL*

Adresavimas

Efektyvus ir absoliutus adresas

Kiekvieną atminties elementą galima adresuoti panaudojant tam skirtus registrus. Išskiriami du adresų atvejai efektyvus ir absoliutus.

Efektyvų adresą (EA) sudaro 4 šešioliktainiai skaitmenys, juo parodomas poslinkis nuo segmento pradžios.

Absoliutų adresą (AA) sudaro 5 šešioliktainiai skaitmenys, juo parodomas unikalus elemento atmintyje adresas. Formuojant absoliutų adresą *dažniausiai* panaudojamas, kuris nors iš segmentinių registrų.

Galioja sąryšis:

*AA = seg * 10h + EA*

kur:

AA – absoliutus adresas (visoje atmintyje)

EA – efektyvus adresas (segmento viduje)

seg – naudojamo segmentinio registro reikšmė

Dažnai šis sąryšis užrašomas per dvitaškį tokioje formoje **seg:EA**

Atvejis	*Segmentas pagal nutylėjimą*	*Ar galima nurodyti kitą segmentą?*	*Efektyviu adresu laikoma*
Komandos paėmimas vykdymui ir procedūros	CS	Ne	IP
Steko viršūnės adresas (PUSH ir POP veiksmams komandose)	SS	Ne	SP
Eilutinės kom. - Šaltinis (source)	DS	Taip	SI
Eilutinės kom. - Gavėjas (destination)	ES	Ne	DI
Operandas atmintyje, kai jo EA suformuoti ***naudojamas* BP (bet kokioje kombinacijoje)**	SS	Taip	Pagal adresacijos baito r/m dalį nurodytas registrų ir poslinkio rinkinys (visų tų registrų laužtiniuose skliaustuose ir poslinkio suma).
Operandas atmintyje, kai jo EA suformuoti ***nenaudojamas* BP (bet kokioje kombinacijoje)**	DS	Taip	
INT tipas pertraukimo procedūros vektorius	Absoliutus adresas yra 4**tipas* (*tipas* – vieno baito dydžio (00...FF ribose esantis) pertraukimo procedūros numeris)		

Ar galima nurodyti kitą segmentą? - jei taip vadinasi naudojant segmento keitimo prefiksą prieš komandos operacijos kodą galima pakeisti numatytąjį segmentą kitu, jei ne segmento keitimo prefiksas tuo atveju komandos vykdymo visiškai neįtakoja.

Operando atmintyje efektyvaus adreso formavimui gali būti panaudoti BX, BP, SI, DI registrų kombinacijos ir poslinkis. Kokia kombinacija naudojama, nurodo adresacijos baito r/m dalis (nebent mod=11, tada operandas yra ne atmintyje, o kažkuris iš procesoriaus registrų).

Komandos paėmimas vykdymui ir procedūros baigęs vykdyti einamąją komandą procesorius eis vykdyti komandos, kurios mašinininis kodas yra absoliučiu adresu ***CS:IP***. *(kita komanda gali būti vykdoma nebūtinai pagal tai, kur einamosios komandos metu rodo CS:IP, jei einamąja komanda vykdomas valdymo perdavimas)*

Segmentas pagal nutylėjimą – parodo, koks segmentas naudojamas, kai nėra jokio jį galinčio pakeisti prefikso, arba prefiksas yra, bet tuo atveju jis neturi įtakos. Taip pat taip skaičiuojami vykdomų procedūrų (CALL,INT adresai, t. y. Per sąryšį CS:IP)

Plėtimo pagal ženklą taisyklė

Baito plėtimas iki žodžio – jei nenurodyta kitaip, vadinasi poslinkis viename baite yra skaičius su ženklu. Tuomet plečiant jį iki dviejų baitų papildomam vyresniame baite reikia surašyti 1tukus arba 0iukus. Priklausomai nuo to koks buvo ženklo bitas (pats pirmas kairėj pusėj).

0000 0001 => 0000 0000 0000 0001
0111 0101 => 0000 0000 0111 0101
1000 0000 => 1111 1111 1000 0000

Šešioliktainė sistema dažniausiai naudojama žodžiams sudėti, arba atimti. Svarbu žinoti, kad norint skaičių turimą viename baite užrašyti tokiu pat skaičiumi žodyje reikia atsižvelgti į ženklą, o kada ne.

Ši taisyklė taikoma tada, kai prie žodinio registro ar efektyvaus adreso tenka pridėti vieno baito poslinkį, tada turime suvienodinti abiejų dėmenų ilgius (baitą išplėsti iki žodžio išlaikant jo ženklą)

Keletas pavyzdžių:

IP=1234h, poslinkis 54h

Tada naujas IP

IP=1234+0054=**1288**h

Tačiau, jei viename baite poslinkis yra 80h arba daugiau, t.y. vyriausias bitas yra vienetas, tada skaičiuojant poslinkį, turime prisirašyti FF (8 vienetinių bitų seka).

Tada naujas IP

IP=1234h+FF80h=1**11B4**h

IP sudaro 4 šešioliktainiai skaitmenys, todėl papildomą vienetą ignoruojame.

Svarbu atsiminti tai, kad išplečiant vieno baito skaičių iki 2 baitų skaičiaus **BŪTINA** atsižvelgti į ženklo bitą:

00h=*0000h*
7Fh=*007Fh*
80h=*FF80h*
FFh=*FFFFh*

Išskyrus išimtinius atvejus, tokius, kaip XLAT komanda (tuomet AL laikomas skaičiumi be ženklo)

Adresacijos baitas

Iškart po OPK kai kuriuose formatuose eina adresacijos (adresavimo) baitas. Po šio dar gali būti 0,1 arba 2 baitai poslinkio. Dažniausiai jis naudojamas, kai kažkuris iš komandos operandų yra registras/atmintis. Jei operandas yra registras arba atmintis, bet ne registras/atmintis tada adresacijos baito paprastai nebūna.

Adresacijos baitas susideda iš:

mod	*reg*	*r/m*
2bit	3bit	3bit

reg laukas gali būti naudojamas ir kaip OPK plėtinys (tuo atveju, kai yra tik vienas operandas r/m)

Ką pasako atskiros adresacijos baito dalys:
mod – modifikatorius

mod	*Prasmė*
00	Po addr. baito eina 0 baitų poslinkis (jokio poslinkio neimama) , išskyrus r/m=110
01	Po addr. baito eina 1 baito poslinkis (jis plečiamas pagal ženklo plėtimo taisyklę iki 2 baitų)
10	Po addr. baito eina 2 baitų poslinkis (pirmiau jaunesnysis baitas, po to vyresnysis baitas)
11	r/m laukas suprantamas, kaip registras, nėra operando atminty

reg – registras
apibrauktoji raudona linija r/m lentelės dalis.

w – width – plotis
word?
0 – ne word => baitas (operandai abu yra baitai)
1 – taip word => žodis (operandai abu yra žodžiai t.y. 2 baitų dydžio)

r/m – register or memory, registras arba atmintis.

Reg arba r/m	**mod=00**	**mod=01, 10**	**mod=11**	
			w=0	**w=1**
000	BX+SI+poslinkis		AL	AX
001	BX+DI+poslinkis		CL	CX
010	BP+SI+poslinkis		DL	DX
011	BP+DI+poslinkis		BL	BX
100	SI+poslinkis		AH	SP
101	DI+poslinkis		CH	BP
110	tiesioginis adresas*	BP+poslinkis	DH	SI
111	BX+poslinkis		BH	DI

***Tiesioginis adresas** – Dviejų baitų poslinkis, be jokių pridėtų registrų.

Taip pat esant operandams: registras, registras/atmintis svarbus d bitas (destination):
Jei *d=0*, tai operandai yra: r/m, **reg**
Jei *d=1*, tai operandai yra: **reg**, r/m
Pirmas operandas vadinamas rezultato, antrasis šaltinio, nes jei turint du operandus kažkuriame saugomas veiksmo rezultatas, tai saugomas pirmąjame (rezultato) operande.

Segmento keitimo prefiksai

Segmento keitimo prefiksai yra 4, kaip ir 4 segmentai.

Visi jie mašininiame kode įrašomi ***prieš operacijos kodą*** (OPK).

Yra ir kitokių tipų prefiksų, kaip, kad REP naudojami eilutinėse komandose, prefiksų tarpusavio eiliškumas nėra apibrėžtas, taip pat šioje architektūroje egzistuoja ir LOCK prefiksas (magistralės blokavimo, bet uždaviniuose jo nebūna).

Reikia mokėti atpažinti segmento keitimo prefiksus iš mašininio kodo fragmento:

Dvejetainė forma: **001xx110**

xx	*segm.*	*Maš. Kodas (šešioliktainėj)*	*Būdas atsiminti eiliškumą*
00	ES	**26**	**E**uropos **S**ąjungoj
01	CS	**2E**	Žaidė **CS**'ą
10	SS	**36**	**S**ovietų **S**ąjungoj
11	DS	**3E**	**D**ėstė **S**aikingai

Stekas

Darbui su steku skirtos dvi komandos.
PUSH ir ***POP*** *(PUSH – įdėti žodį į steką, POP – paimti žodį iš steko)*
SF registro keitimui ir pasidėjimui į steką naudojamos atskiros asemblerinės mnemonikos
PUSHF (Push Flags) reiškianti PUSH SF
POPF (Pop Flags) reiškianti POP SF

SS registras rodo į steko segmento pradžią (skaičius nurodytas paragrafais, skaičiuojant absoliutų adresą dauginamas iš 10h). SP registras rodo poslinkį nuo steko segmento pradžios. SS:SP sąryšio jokie segmento keitimo prefiksai neįtakoja.

Naudojant **PUSH** komandą žodinio registro turinys, arba 2 baitai iš nurodytos atminties vietos yra įdedami į steką, prieš tai SP sumažinus 2 vienetais. (viršijus FFFF vėl skaičiuojama nuo 0000)
Naudojant **POP** komandą 2 baitai iš steko patalpinami į nurodytą atminties vietą arba į žodinį registrą, po to SP padidinamas 2 vienetais. (peržengus 0000 vėl skaičiuojama nuo FFFF stekas yra ciklinis)

Komandų atliekami veiksmai:
PUSH:
1. SP:=SP-2; (sekančiuose veiksmuose naudojamas naujai gautas SP)
2. mov SS:SP, jaunesnysis baitas (registro, arba pirmas pirmas baitas iš atminties)
   mov SS:(SP+1), vyresnysis baitas (registro, arba sekantis po jaunesniojo iš atminties)

POP:
1. mov jaunesnysis baitas, SS:SP
   mov vyresnysis baitas, SS:(SP+1)
2. SP:=SP+2;

Pavyzdys:
(kad ir h raidės nėra, tai vis tiek šiuo atveju, kaip ir egzamino užduotyse reikš šešioliktainius skaičius)
Turime SP=1234;
AX=75F4; BX=7894; CX=DEAD;

Pradinė steko būsena:

SP	*Baito reikšmė*
1230	05
1231	40
1232	11
1233	36
1234	7C

>>>>>>

Atliekam veiksmą PUSH AX.
Naujas SP=1232
AX, BX, CX nesikeičia

SP	*Baito reikšmė*
1230	05
1231	40
1232	***F4***
1233	***75***
1234	7C

*Po to atlikę **POP** CX pakeistume CX, SP reikšmes, bet ne steko turinį – atliekamas tik nuskaitymas (dar keisis IP registras, jis vykdant komandas perskaičiuojamas visuomet). Tada vėl būtų SP=1234h, o nauja CX reikšmė CX=75F4*

Pavyzdys parodytas su registro pasidėjimu į steką (PUSH), iš tikrųjų galima pasidėti ir kokį nors elementą iš atminties. Tuomet ***segmentas:EA*** rastas baitas yra laikomas jaunesniuoju, ***segmentas:(EA+1)*** baitas laikomas vyresniuoju. Tas pats galioja ir pasiimant žodį iš steko (POP).

Status Flag registras (SF)

Kas yra SF registras?

Atliekant įvairias komandas visuomet keičiama IP registro reikšmė. Tačiau atskirais atvejais (ypač atliekant aritmetines ir logines komandas) keičiamas Status Flag registras. Atskiri jo bitai nurodo tam tikrus požymius. Šis registras dar vadinamas „procesoriaus būsenos registru“.

Registras yra žodžio (16 bitų) dydžio.

Reikia žinoti šių bitų eiliškumą, ir kokia kiekvieno iš jų prasmė (neužpildyti langeliai reiškia, kad tie bitai yra nenaudojami jokiam požymiui saugoti, vykdant komandas yra nekeičiami).

Išskyrus POPF komandą, kuomet iš steko paimama nauja SF registro reikšmė, iš naujo nustatomi visi bitai, net ir nenaudojami). Taip pat SF reikšmės paėmimas vykdomas ir baigus vykdyti pertraukimo apdorojimo programą (su IRET komanda).

SF bitų reikšmės

Kiekvienas konkretus bitas vadinamas „flagu“, gali turėti reikšmę 0 arba 1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	**DF**	**IF**	**TF**	**SF**	**ZF**		**AF**		**PF**		**CF**

Rezultato laukas – tai yra rezultatas atlikus loginį arba aritmetinį veiksmą su dviem operandais. Rezultato laukas yra lygiai tokio paties dydžio, kaip ir operandai, jei veiksmai atliekami su žodžiais, tai rezultato laikas yra žodis, jei baitais – baitas. Carry Flag'o bitas NEPRIKLAUSO rezultato laukui.

Žaliai pažymėtieji bitai kontroliuoja procesoriaus būseną. Vykdant aritmetines/logines komandas jie nesikeičia.

Kam jie naudojami ir ką jie reiškia:

Flagas	Pavadinimas	Naudojimas ir reikšmė
CF	*Carry Flag* „pasiskolinimo/pernešimo požymis“ (skaičių be ženklo perpildymo požymis)	Aritmetinėse operacijose atstoja papildomą bitą rašomą kairiau vyriausiojo rezultato lauko baito. Baitų/žodžių sudėties atveju reiškia papildomą už rezultato lauko išeinantį bitą „mintyj“, atimties atveju reiškia pasiskolinimą iš už rezultato lauko ribų. Galima suprasti ir kitaip. Jei abu skaičiai yra skaičiai be ženklo iš intervalo [0;255] dirbant su baitais ir [0;65535] dirbant su žodžiais. Tai rezultatui nebetelpant tame intervale CF=1, kitu atveju CF=0 Atliekant logines komandas veikiančias kiekvienam bitui atskirai (AND, OR, XOR) CF=0 **Tas papildomas CF bitas reikalingas tik siekiant gauti CF flago reikšmę. Apskaičiuojant kitus flagus į jį nežiūrima.**
PF	*Parity Flag* „lyginumo požymis“	Ar vienetinių bitų kiekis rezultato lauko jauniausiame baite (žiūrimi jauniausi **aštuoni(!!!)** bitai) yra lyginis? (jei jų nėra nė vieno, laikoma, kad lyginis, nes gi nulis yra lyginis skaičius). Jei taip, PF=1 Jei ne PF=0
AF	*Auxilliary Carry Flag* „papildomas pasiskolinimo/pernešimo požymis“	Reiškia pasiskolinimą arba pernešimą iš jauniausio į vyresnįjį pusbaitį. Jei pernešimas arba pasiskolinimas toje vietoje įvyksta AF=1 Jei ne AF=0
ZF	*Zero Flag* „nulio požymis“	Ar rezultato laukas yra sudarytas vien iš nulinių bitų? Taip, ZF=1

		Ne, ZF=0
SF	*Sign Flag* „ženklo požymis“	Vyriausio rezultato lauko bito reikšmė (skaičiuose su ženklu 1 reiškia minusą, 0 pliusą)
TF	*Trap Flag* „spąstų požymis“	Parodo, ar kaskart įvykdžius komandą įvyksta žingsninis pertraukimas (INT 1). Naudojamas debuginimui. Jei taip, TF=1 Jei ne, TF=0
IF	*Interrupt Flag* „pertraukimo leidimo požymis“	Ar leidžiami išoriniai maskuojami pertraukimai Jei taip, IF=1 Jei ne, IF=0
DF	*Direction Flag* „krypties požymis“	Parodo kaip vykdant eilutines komandas keičiasi registrai SI, DI (abu arba vienas iš jų, priklausomai nuo komandos). Jei DF=0, eilutinėje komandoje panaudoti (SI, DI arba abu) registrai yra didinami Jei DF=1, mažinami
OF	*Overflow Flag* „perpildymo požymis“ (skaičių su ženklu perpildymo požymis)	**Aritmetinėse komandose** atliekant sudėtį arba atimtį operandai ir rezultatas yra suprantami kaip skaičiai priklausantys skaičiaus su ženklu intervalui. Vieno baito atveju [-128; +127] Vieno žodžio atveju [-32768; +32767] Sudėję arba atėmę turimus SKAIČIUS SU ŽENKLU (jei tokie neduoti turime į tokius pasiversti) žiūrim ar gautas rezultatas yra teisingai gautas skaičius su ženklu, jei ne tada OF=1. **Postūmių komandose**, pastumiant per vieną bitą pasako ar keičiasi ženklo bitas, prieš postūmį ir po jo, jei taip, OF=1, jei ne OF=0. Jei postūmis per daugiau bitų OF neapibrėžtas. **Alikus loginę komandą** veikiančią kiekvienam bitui atskirai (AND, OR, XOR) visuomet OF=0

Aritmetinėse komandose **CF** galima suprasti, kaip „perpildymą be ženklo“ ir **OF** „perpildymą su ženklu“ Norint suprasti šiuos dalykus, reikia mokėti bet kokį skaičių pasiversti į skaičių be ženklo, jei toks neduotas, arba į skaičių su ženklu, priklausomai nuo to, kas skaičiuojama.

PASTABA: *Kad ir kokia didelė pagunda būtų atimtį keisti sudėtimi pakeičiant antro skaičiaus (to, kuris yra atimamas) ženklą, to daryti negalima, nes tada dalis flagų gaunami ne tokie, kokie priklauso.*

OF požymio nustatymas

Baituose atliekame dviejų dešimtainių skaičių sudėtį 14 + 27:

Imame *14* → teigiamas skaičius, patenka į skaičiaus su ženklu intervalą *[-128;127]*, turime, kad:

14_{10} = 0000 1110

Tas pats ir su 27, turime:

27_{10} = 0001 1011

Paskaičiuojame:

+	0000 1110 0001 1011
0	0010 1001

Kai ženklo bitas nulinis, turime teigiamą skaičių, vadinasi nereikės atlikti jokių papildomų

invertavimų.

Tiesiog nustatome, kad:
$0000\ 1110 = 14_{10}$
$0001\ 1011 = 27_{10}$
$0010\ 1001 = 41_{10}$

Patikriname, ar teisinga lygybė 14 + 27 = 41.
Taip.
Perpildymo nebuvo, OF=0.

Baituose atliekame dviejų dešimtainių skaičių sudėtį: (-127)+129, norime rasti OF:

-127<0, tai gauname +127 ir pakeičiame ženklą (t. y. bitų inversija ir +1)
$+127 = 0111\ 1111_2$ => invertavus turime 1000 0000, dar atliekame +1 => 1000 0001
Vadinasi -127 dvejetainėje yra 1000 0001

129>0, tai dvejetainį užrašą gauname iškart 128+1 => 1000 0001

Atliekame sudėtį:

+	1000 0001 1000 0001
1	0000 0010

OF požymis pasako, ar įvyko perpildymas skaičiuose **su** ženklu.
Tai patikrinti galima taip: atliekame aritmetinį veiksmą su abiem dėmenimis ir rezultatu kaip skaičiais su ženklu ir žiūrime, ar gavosi teisinga lygybė:
Jei teisinga, skaičiaus su ženklo perpildymo nebuvo → OF=0
Jei neteisinga, įvyko skaičiaus su ženklu perpildymas → OF=1

Pirmas dėmuo: 1000 0001, ženklo bitas 1 todėl pirmiausia pakeičiame ženklą:
invertavę turime 0111 1110, dar pridėję +1 turime 0111 1111 (tai yra +127, todėl pradinis prieš pakeičiant ženklą buvo -127)
Antras dėmuo: (tas pats kas su pirmu) → -127

Rezultatas: ženklo bitas 0, iškart turime teigiamą, nustatėme, kad tai yra 2.

Pasitikriname:
(-127) + (-127) = 2 ?
Ne, todėl OF=1.

Pertraukimai

Kas tai yra ir kokios jų rūšys

Pertraukimu vadinamas procedūros iškvietimas (valdymo perdavimas) panaudojant komanda:
INT n.

Pačios *INT n* komandos kode gali ir nebūti, tada programos vykdymo metu operacinė sistema suspėja įvykdyti tam tikrą kiekį pertraukimų. Labiausiai pertraukimų nauda atsispindi programuojant, tuomet pertraukimų procedūra vietoj to, kad rašyti ir taip dažnai visose programose naudojamą procedūrą (tarkim simbolio išvedimas į ekraną), atliekamas pertraukimo procedūros iškvietimas, ji įvykdoma ir valdymas grįžta pagrindinei pertraukimą iškvietusiai lyg to pertraukimo išvis nebūtų įvykę, išskyrus atvejus kuomet pertraukimo procedūra grąžina savo vykdymo rezultatus pagrindinei programai (per registrus), tarkim atidarant failą į BX įrašomas failo deskriptorius ir pan.

Instrukcijos pertraukimo procedūroms perduodamos per įvairių registrų reikšmes. INT n, *n procedūros numeris* parodo tik kurią pertraukimo procedūrą iškviesti, o registrų reikšmes interpretuoja pati procedūra, pagal tai kaip yra suprogramuota, tarkim iškvietus INT 21h, tikrinama AH registro reikšmė, kuria nusakoma reikalinga funkcija (spausdinti simbolį, atidaryti failą ir kt.).

Bet koks pertraukimas gali būti įvykdytas tik tada, kai baigta vykdyti tuo metu vykdoma komanda.

Iš pertraukimo procedūros grįžtama naudojant valdymo perdavimo komandą IRET, ji visada turi būti pertraukimo procedūros pabaigoje.

Skirstymas:

Pertraukimai procesoriaus atžvilgiu skirstomi į vidinius ir išorinius, išoriniai gali būti maskuojami arba nemaskuojami.

Vidiniai pertraukimai kyla procesoriaus darbo metu ir negali būti uždrausti, išskyrus žingsninį vidinį pertraukimą reguliuojamą per TF požymį SF registre (pavyzdžiui skaičiuojant įvyko dalyba iš nulio).

Išoriniai nemaskuojami pertraukimai kyla ne pačiame procesoriuje, bet dėl tam tikrų išorinių priežasčių negali būti jo ignoruojami (pavyzdžiui nutrūko elektros energijos tiekimas kompiuteriui).

Išoriniai maskuojami pertraukimai kaip ir nemaskuojami kyla ne pačiame procesoriuje, tačiau jie priklausomai nuo IF požymio SF registre gali būti vykdomi arba ignoruojami „yra galimybė juos užmaskuoti“ (pavyzdžiui klaviatūra renkamas tekstas, judinama pelė).

Išimtis:

Jei vykdant komandą yra keičiamas segmentinis registras
mov seg.reg, ...
pop seg.reg
tai siekiant išvengti neapibrėžtumų pertraukimas įvyks tik po dar vienos įvykdytos komandos.
Kitu atveju kilęs pertraukimas bus įvykdytas iškart po to, kai bus įvykdyta dabar vykdoma komanda.

Pagrindiniai pertraukimų tipai ir jų prioritetai

Procedūros INT **n** priklausomai nuo skaičiaus n reikšmės dar turi ir savo pavadinimus, bei situacijas kuomet yra automatiškai iškviečiami:

***n** reikšmė*	*Pavadinimas*	*Kada įvyksta**
0	*Dalyba iš nulio*	Dalybos perpildymas kilęs vykdant DIV arba IDIV komandą
1	*Žingsninis rėžimas*	Po kiekvienos komandos, jei flagas TF=1
2	*Nemaskuojamas išorinis*	Esant nemaskuojamam išoriniam pertraukimui
3	*Kontrolinis taškas „breakpoint“*	Sutikus komandą maš. kodu CC. Naudojama debugeriuose. Asemblerinė mnemonika *INT 3*
4	*Perpildymo apdorojimas naudojant komandą INTO*	Kai programoje vykdoma komanda INTO ***ir*** flagas OF=1
5-31	*OS reikmėms*	Priklauso nuo operacinės sistemos, įvyksta iškvietus atitinkamą procedūrą, **svarbu: pagrindinis operacinių sistemų (Windows, Linux ir kt.) yra būtent pertraukimų apdorojimo procedūrų skirtumai. Windows ir Linux turi savo, kokius turi Linux galite pažiūrėti paieškoję „POSIX system calls“**
32-255	*Naudotojo reikmėms*	Suprogramuojama kompiuterio vartotojo naudojamos programinės įrangos (pvz. koks nors „Skype“ gali turėti savo interuptų), įvyksta iškvietus atitinkamą procedūrą

* - Programos kode iškviečiant pertraukimą tarkim INT 1, jis įvyksta nepaisant jokių sąlygų.
Jei kviečiama komanda yra INTO, tai pažiūrima ar OF=1 (flagas), jei taip įvykdoma INT 4 komanda, jei ne jokie veiksmai šia komanda neatliekami.

Prioritetas – pirmenybė.
Kai vykdant programą įvyksta keli pertraukimai, tai egzistuoja prioritetai, pagal kuriuos, vykdomas žemesnio prioriteto pertraukimas gali būti pats pertrauktas aukštesnio prioriteto, tokiu atveju žemiausias prioritetas teikiamas pagrindinei programai, kuri gali būti pertraukta, bet kurio pertraukimo.

1. Dalyba iš nulio
2. INT n komanda pačioje programoje
3. INTO komanda
4. Nemaskuojamas išorinis
5. Maskuojamas išorinis
6. Žingsninis, reguliuojamas per Trap Flag.

Pertraukimų vykdymas ir vektorių lentelė

Pertraukimas iškviečiamas valdymo perdavimo komanda INT n, kur n yra pertraukimo numeris.
Į pagrindinę programą iš pertraukimo grįžtama naudojant valdymo perdavimo komandą IRET.
Apie jų atliekamus veiksmus aprašyta valdymo perdavimo skyriuje.

Šiame skyrelyje plačiau apžvelgiama, komandos INT n vykdymas.
Į steką paeiliui padedami SF, CS, IP registrai.
TF ir IF flagai nustatomi nuliniais, tam, kad pertraukimo vykdymo metu nekiltų kokių kitų žingsninių ir būtų neleidžiami išoriniai maskuojami pertraukimai.
Valdymo perdavimui CS ir IP reikšmės imamos iš vektorių lentelės.

Vektorių lentelė – pertraukimų procedūrų adresai išreikšti poromis CS:IP, sudėti paeiliui kiekvienam pertraukimui.
Pertraukimų numerių yra 256 [0;255], o vienas CS ir IP rinkinys užima 4 baitus.
Pradedant absoliučiu adresu 00000 pirmasis kilobaitas yra užpildytas vektorių lentelės duomenimis.
Tarkim, kad įvyko n-tasis pertraukimas, tada į steką patalpinus SF,CS,IP, flagus IF ir TF nustačius nuliais naujos CS ir IP reikšmės imamos iš vektorių lentelės pradedant adresu 4n tokia tvarka:

4n	*IP jaunesnysis baitas*
4n+1	*IP vyresnysis baitas*
4n+2	*CS jaunesnysis baitas*
4n+3	*CS vyresnysis baitas*

Vadinasi įvykdžius INT n komanda valdymas perduodamas absoliučiu adresu CS:IP, kur abiejų registrų reikšmės imamos iš vektorių lentelės.

Įvykdžius komandą IRET, valdymas grąžinamas pertraukimą iškvietusiai programai iš steko paimant IP, CS, SF reikšmes.

Pavyzdžiui vektorių lentelėje nuo 4n turime baitus 12 34 56 78, tada naujos registrų reikšmės bus:
CS=7856
IP=3412
Beje, pateikti baitai 12 34 56 78 yra šešioliktainėje, ne dešimtainėje sistemoje.
Turėdami dešimtaines reikšmes turite pasiversti į šešioliktaines.

Darbas su įvairaus formato skaičiais

Koprocesoriaus (8087) realiųjų skaičių formatai (float'ai)

8087 koprocesoriuje naudojami 3 pagrindiniai realiųjų skaičių formatai. (*IEEE-754* standartas)

Trumpas realus (**keturiuose** baituose)

S (1b)	*charakteristika (8b)*	*mantisė (23b)*

- 1 ženklo bitas (0 reiškia pliusą, 1 reiškia minusą)
- 8 bitai charakteristika (eilė + 127_{10}) 127_{10}=**7Fh**

  Eilė – dvejeto laipsnis, kai skaičius užrašomas normalizuota forma (dešimtainėje populiariau vadinama standartine skaičiaus išraiška). T.y. sveikas skaičius, rodantis kokiu laipsniu reikia pakelti dvejetą, kad padauginus *1,mantisė* iš $2^{eilė}$ gautume tą patį skaičių.
- 23 bitų mantisė
- Skaičių būtina privesti prie normalizuotos formos, kuri atrodo taip:

$$(-1)^s * 2^{eilė} * 1,\text{mantisė}$$

Ilgas realus (**aštuoniuose** baituose)

S(1b)	*charakteristika (11b)*	*mantisė (52b)*

- 1 ženklo bitas (0 reiškia pliusą, 1 reiškia minusą)
- 11 bitų chrakteristika (eilė + 1023_{10}) 1023_{10} = **3FFh**
- 52 bitų mantisė
- Skaičių būtina privesti prie normalizuotos formos, kuri atrodo taip:

$$(-1)^s * 2^{eilė} * 1,\text{mantisė}$$

Vidinis realus (**dešimtyje** baitų)

S (1b)	*charakteristika (15b)*	*i bitas (1b)*	*mantisė (63b)*

- 1 ženklo bitas (0 reiškia pliusą, 1 reiškia minusą)
- 15 bitų charakteristika (eilė+16383_{10}) 16383_{10} = **3FFFh**
- 1 i bitas, jis parodo koks skaičiukas yra prieš kablelį po kurio eina mantisė (šiame formate nebūtina sudaryti normalizuotos formos, bet patartina tai padaryti ir i bitą visuomet žymėti vienetu)
- 63 bitų mantisė
- Normalizuota forma atrodytų taip: $(-1)^s * 2^{eilė} * 1,\text{mantisė (i bitas=1)}$
- ***Ne***normalizuota forma: $(-1)^s * 2^{eilė} * i,\text{mantisė}$

Pastaba: Užtenka prisiminti normalizuotos formos užrašą kiek užima charakteristika, ir ar yra **i** bitas (jis yra tik 10ties baitų atveju). Tada tokį eiliškumą:

1. Ženklo bitas (visada tik vienas bitas)
2. Charakteristika, kuri esant nulinei eilei yra lygi: kairiausias bitas nulis visi kiti vienetai.
3. i bitas, jei šis yra (tik dešimties baitų atveju)
4. mantisė eina iki pat galo.

Mantisė yra nukerpama ir kas netilpo tiesiog neįsimenama. Iš tikrųjų kompiuteriuose esant bitui už mantisės lygiam vienetui vyksta apvalinimas (mantisė+1), apvalinant perpildžius mantisę vienu padidėja charakteristika, tačiau to žinoti nebūtina (galima ir neapvalinti nesukant sau galvos).
Sudarinėjant normalizuotą formą kai yra slankiojamas kablelis tai slenkant jį į kairę pusę charakteristika (ir eilė) didinamos, į dešinę – mažinamos. Šiame konspekte sprendžiamuose uždaviniuose laikysime, kad mantisė yra tiesiog nukerpama.
***Išimtis, kurią vertėtų žinoti** – jei charakteristika ir mantisė yra užildytos tik nuliais, tai skaičius yra nulis.*

Dešimtainio slankaus kablelio vertimas į šešioliktainį užrašą

Šis pavyzdys parodo kaip **keturiuose baituose** užrašyti dešimtainį skaičių ***165,29*** panašios taisyklės taikomos ir kitiems formatams (juose arba yra i bitas, arba tiesiog ilgesnė mantisė ir charakteristika saugoma didesniame bitų skaičiuje).

Versdami su sveikąja ir trupmenine dalimis dirbame atskirai

SVEIKOJI DALIS:

- Pasiverčiame skaičių 165 į dvejetainę skaičiavimo sistemą $165_{10}=1010\ 0101_2$
- Po vyriausio vieneto matome einančius 7 bitus, todėl jau 7 mantisės bitus jau turime.
- Reikia likusių 23-7=16-ikos

TRUPMENINĖ DALIS:

0,29 yra skaičiaus trupmeninė dalis

Daugindami *2 imdami sveikąją dalį (rašysim mantisėj), o toliau vėl dirbdami tik su trupmenine dalimi tol kol rasime reikiamus 16 skaičiukų:

0,29 * 2 = **0**,58
0,58 * 2 = **1**,16
0,16 * 2 = **0**,32
0,32 * 2 = **0**,64

0,64 * 2 = **1**,28
0,28 * 2 = **0**,56
0,56 * 2 = **1**,12
0,12 * 2 = **0**,24

0,24 * 2 = **0**,48
0,48 * 2 = **0**,96
0,96 * 2 = **1**,92
0,92 * 2 = **1**,84

0,84 * 2 = **1**,68
0,68 * 2 = **1**,36
0,36 * 2 = **0**,72
0,72 * 2 = **1**,44

Vis tik yra ir kitas būdas surasti šiuos skaitmenis. Atliekame tą patį, tik vietoj 4 daugybos iš 2 veiksmų po vieną daugybos iš 16 veiksmą. Gautą sveikąją dalį užrašome kaip šešioliktainį skaičių, o rašant į mantisę išsiskleidžiame kiekvieną šešioliktainį skaitmenį iki keturių dvejetainių skaitmenų.

Daugybą tuomet tikslingiau atlikti ne mintinai, o stulpeliu.

Pvz. galėjome šiuo atveju daryti ir taip:

0,29 * 16 = **4**,64 (4=**0100**)
0,64 * 16 = **10**,24 (10=**1010**)
0,24 * 16 = **3**,84 (3=**0011**)
0,84 * 16 = **13**,44 (13=**1101**)

Gavome tuos pačius skaičius, per mažiau veiksmų, tačiau tam reikia mokėti greitai paversti skaičius nuo 1 iki 15 į keturis dvejetainius skaitmenis.

165,29 skaičius dvejetainiame pavidale (ne tikslus, o tik tiek, kiek telpa pagal formatą):
1010 0101, 0100 1010 0011 1101

Taigi turime:

ženklo bitas 0, nes skaičius teigiamas
eilė = +7, nes stumiame kablelį per 7 vietas į kairę, kad gautume normalizuotą formą
charakteristika = eilė + 127 = 7 + 127 = 134 (128+4+2, t.y. 1000 0110)
mantisė pateikta geltonam fone

sudarome atsakymą:
antroje eilutėje surašyti šešioliktainiai atitikmenys, kuriuos rašome į atsakymą

0100	***0011***	***0010***	***0101***	***0100***	***1010***	***0011***	***1101***
4	***3***	***2***	***5***	***4***	***A***	***3***	***D***

Atsakymas: 43 25 4A 3D

Šešioliktainio realaus skaičiaus vertimas į dešimtainį užrašą

Darome iš kito galo.
Paimame prieš tai gautą 43 25 4A 3D.
Užsirašome į dvejetainę:

0*100 0011 0*010 0101 0100 1010 0011 1101

Iš to, kad užima 4 baitus ir yra *realus* skaičius atpažįstame, kad tai yra *trumpas realus* formatas.
Žinodami formatą, žinome ir kur yra **ženklo bitas**, *charakteristika* ir mantisė.
Iš jų nustatome, kad ženklo bitas yra 0 (skaičius teigiamas), charakteristika 1000 0110 = 134, t. y.
Eilė yra 127 + eilė = 134, vadinasi eilė = 7.
Mantisė 010 0101 0100 1010 0011 1101 ir reikalingas vienetukas prieš ją. Todėl, turime
1,010 0101 0100 1010 0011 1101

Iš ko galiausiai gauname, kad tai yra:
+1,010 0101 0100 1010 0011 1101 * 2^7 = + **110 0101 0**,0100 1010 0011 1101 = **165**,(kažkiek).

Su trumpmenine dalim padarom taip: padarom trupmeną, paimame kiek skaitmenų eina po kablelio... ? 16. Vadinasi vardiklis bus 2^{16} o skaitiklis 0100 1010 0011 1101 = 4A3D = 19005.
T.y. 19005 / 2^{16} = 0.2899932861328125

Gauname, kad rezultatas yra 165,2899932861328125

Tai skiriasi nuo pradinio ***165.29***, nes skaičių užrašant realiaisiais formatais dalis tikslumo prarandama, bet tarkim šiuo atveju jei tikslumas domina tik šimtųjų atžvilgiu – suapvalinę pradinį skaičių ir gautume. Tikslumas prarandamas, bet gana nežymiai, šiuo atveju tik per 165.29 – 165.2899932861328125 = 0.0000067138671875 kas yra ne toks jau žymus skirtumas viso skaičiaus atžvilgiu. Norėdami dar didesnio tikslumo galėtume naudoti ilgą realų arba vidinį realų formatus.

Supakuoti ir išpakuoti skaičiai

Kartais prireikia saugoti skaičius baituose dešimtainėje sistemoje. Tam naudojami išpakuoti ir supakuoti skaičiai.

Išpakuoti skaičiai – tai skaičiai, kur viename baite gali būti saugomos reikšmės tik nuo 0 iki 9.
nuo 0000 0000 iki 0000 1001

Supakuoti skaičiai – taupant atmintį skaičiai pakuojami.
Tuomet pusbaičiuose galima saugoti skaičius nuo 0000 iki 1001.
t.y. Galimos reikšmės nuo 00h iki 99h (visos reikšmės su raidėmis laikomos neteisingais supakuotais skaičiais)

Supakuoti skaičiai naudojami DECIMAL formate. T.y. Stabilaus (fiksuoto) kablelio formatas, jame prieš panaudojimą numatoma, keli skaitmenys skiriami sveikąjai daliai, keli trupmeninei.
Pavyzdys:
Tarkim turim formatą, kurį sudaro 8 skaitmenys, iš kurių 6 yra sveikoji dalis (turimai pinigų sumai saugoti ir pan.)
Duoti baitai: 10 24 39 17
Tada šį skaičių galime užrašyti realiuoju taip: 102439,17

Baitas ir pusbaičiai suprantami taip:

Baitas	
Vyresnysis pusbaitis	Jaunesnysis pusbaitis

Pusbaitį sudaro 4 bitai, pusbaitis gali įgyti 16 skirtingų reikšmių iš intervalo [0;15].
Pusbaičio reikšmė užrašoma vienu šešioliktainiu skaitmeniu.

Darbui su supakuotais ir išpakuotais skaičiais egzistuoja 6 komandos (be operandų), vardai susideda iš trijų raidžių:
AAA, AAS, AAM, AAD
DAA, DAS

Ką pasako komandos pavadinimas:

Pirmoji raidė:
A – dirbama su išpakuotais skaičiais (nuo žodžio **A**scii)
D – dirbama su supakuotais skaičiais (nuo žodžio **D**ecimal)

Antroji raidė visuomet **A** *(reiškia žodį Adjust)*

Trečioji raidė:
A – **A**ddition - sudėtis
S – **S**ubtraction – atimtis
M – **M**ultiplication – daugyba
D – **D**ivision – dalyba

Korekcijos dalybai ir daugybai atliekamos tik su išpakuotais skaičiais.
Visos komandos yra vykdomos po to, kai AL arba AX registre turime suformuotą atsakymą – išimtis dalybos komanda (AAD), vykdoma prieš dalybos veiksmą.

(AL and 0Fh)>9 reiškia „AL jaunesnis pusbaitis yra didesnis už 9“, t.y. atliekame tokią operaciją ???? ???? and 00001111 => 0000???? ir gautą skaičių lyginame su 9. AND veiksmas vykdomas kiekvienam bitui atskirai.

Komandos darbui su išpakuotais skaičiais

Išpakuotų skaičių korekcijoms egzistuoja 4 komandos.

AAA ir AAS komandos:

AAA komandos algoritmo kodas yra toks:

if*((AL and 0Fh) >9) or (AF=1)) **then***
AL := AL + 6
AH:= AH + 1
AF:= 1
CF:= 1
else
AF:=0
CF:=0
endif
AL:=AL and 0Fh

AAS komandos algoritmas gaunamas **raudonus pliusus** pakeitus minusais.

AAA algoritmas žodžiais:

- Tikrinama ar AL jaunesnysis pusbaitis viršija 9 ARBA AF flag'as tapo vienetu (įvyko pernešimas į vyresnįjį pusbaitį). **Jei bent viena sąlyga išsipildo**, tada:
  - AL padidinamas 6-iais
  - AH padidinamas vienetu
  - AF ir CF flag'ai nustatomi 1
- **Jei nė viena sąlyga neišsipildė**
  - AF ir CF flagai nunulinami.
- Atlikus visus veiksmus vyresnysis AL pusbaitis išvalomas (padaromas nuliumi, pvz baitas 54h taptų 04h)

AAS algoritmo atveju žodis padidinamas keičiamas žodžiu sumažinamas.

AAM algoritmas:

AH := AL div 10_{10}
AL := AL mod 10_{10}

T.y. AL padalinamas iš 10_{10}, sveikoji dalis išsaugoma AH registre, liekana AL registre.

AAD algoritmas:

AL := AH * 10_{10} + AL
AH := 0

Pastaba: AAM ir AAD naudojamas dešimtukas yra 10 dešimtainėje sistemoje (t. y. 0A šešioliktainėje)

Komandų panaudojimo tikslai:

- Komandos AAA, AAS, AAM vykdomos **tik tada, kai** sudėjus/atėmus/sudauginus (priklausomai nuo situacijos) 2 išpakuotus dešimtainius skaičius rezultatą turime registre AL, komandos sutvarko rezultatą į taisyklingą dešimtainį išpakuotą skaičių (po komandos įvykdymo rezultatas išsaugomas AX registre).
- Komanda AAD vykdoma **prieš tai**, kai norima atlikti dalybos veiksmą su išpakuotais dešimtainiams skaičiams.

Komandos darbui su supakuotais skaičiais

Su supakuotais skaičiais galima atlikti tik sudėties ir atimties rezultatų korekcijas.
Tam yra komandos DAA ir DAS

DAA komandos algoritmo kodas yra toks:

if*(((AL and 0Fh) >9) or (AF = 1))* ***then***
    AL := AL + 6
    AF := 1
endif

if*((AL>9Fh) or (CF=1))* ***then***
    AL :=AL+ 60h
    CF:=1
endif

Komandos **DAS** atveju, vietoj **raudonų pliusų** yra minusai.

DAA Algoritmas žodžiais:

- Jei AL jaunesnysis pusbaitis viršija 9 ARBA flagas AF=1,
  - Jei bent viena sąlyga pasitvirtina tada prie AL ***pridedam*** 6, nustatom, kad AF=1. Pereinam prie kito punkto.
  - Jei abi sąlygos klaidingos iškart pereiti prie sekančio punkto.
- Tikrinama ar AL reikšmė viršija 9Fh ARBA CF=1.
  - jei bent viena sąlyga pasitvirtina prie AL ***pridedam*** 60h, ir nustatom, kad CF=1.
  - Jei abi sąlygos klaidingos šį punktą praleidžiam, lyg jo nebūtų

Komandas DAS atveju vietoj žodžių „pridedam“ yra žodžiai „atimam“.

Komandų panaudojimo tikslai:

- DAA ir DAS naudojami po to, kai buvo atlikta sudėties arba atimties operacija su dviem supakuotais skaičiais. Komanda koreaguojamas AX registre esantis sudėties arba atimties rezultatas į teisingą supakuotą skaičių.

Kiti koprocesoriaus skaičių formatai

1. **Sveikas žodis (16bitų), dvejetainiu papildomu kodu** – dviejų baitų skaičius su ženklu. „Papildomu kodu“ čia turima minty, kad neigiamos reikšmės saugomos jas invertavus ir pridėjus vienetą, o vyriausias bitas reiškia ženklą.
   Pavyzdžiui:
   -1: FFFF
   0: 0000
   1: 0001

2. **Sveikas trumpas (32bitai), dvejetainiu papildomu kodu** – keturių baitų skaičius su ženklu.
   Pavyzdžiui:
   -1: FFFFFFFF
   0: 00000000
   1: 00000001

3. **Sveikas ilgas (64bitai), dvejetainiu papildomu kodu** – aštuonių baitų skaičius su ženklu.
   Pavyzdžiui:
   -1: FFFFFFFFFFFFFFFF
   0: 0000000000000000
   1: 0000000000000001

4. **Supakuotas dešimtainis išplėstas (80bitų):**

S (1b)	*X (7b)*	*Skaitmenys (72b)*

80bitų:

s – ženklo bitas (1 – jei skaičius neigiamas, 0 – jei teigiamas)

x – 7bitai su neapibrėžtomis reikšmėmis, neturintys įtakos (atliekant reikšmės rašymą į atmintį jie būna nuliniai)

Skaitmenys – 72 bitai, po 4bitus kiekvienam 0-9 skaitmeniui užkoduoti.

Pavyzdžiui:

-1: 80000000000000000001

0: 00000000000000000000

1: 00000000000000000001

-15_{10}: 80000000000000000015

Komandų klasifikacija ir jų atliekami veiksmai

Valdymo perdavimo komandos

Bendra informacija

Valdymo perdavimo komandos yra skirtos valdymo perdavimui kitu adresu. Atliekant valdymo perdavimą kodas toliau vykdomas ne einantis atmintyje po einamosios komandos, o iš bet kurios kitos nurodytos atminties vietos.

Tai naudinga kuriant programas, kuriose reikalingos procedūros, ciklai ir pan. Taip pat vykdant kodą dažnai tenka sureaguoti į tam tikras sąlygas (tarkim, jei rezultatas buvo nulinis, tada išspausdinti atitinkamą pranešimą ir pan).

Valdymo perdavimo komandos skirstomos į **sąlygines** ir **besąlygines**, priklausomai nuo to, ar jas vykdant tikrinama kokia nors sąlyga (CX registro reikšmė, arba tam tikri SF registro flagai).

IP registro reikšmė komandos vykdymo metu

Komandos vykdymo metu IP registras rodo į sekančios vykdomos komandos pradžią. Pavyzdžiui, jei turime tokį kodo segmento fragmentą:

2657: EB AA 96 74 (2657 – poslinkis kodo segmente).

Tai reikia žinoti, kad vykdant komandą IP registras rodys ne į dabar vykdomos, o į sekančios vykdomos komandos pradžią. Tai labai svarbu tada, kai tenka pridėti poslinkį (tuomet reikia žinoti prie ko jį reikia pridėti).

Šiuo konkrečiu atveju komandos ilgis yra 2 baitai EB AA, todėl komandos vykdymo metu IP bus 2657+2=2659 (šešioliktainiai skaitmenys!!!).

Taip yra todėl, nes vykdant einamąją komandą procesorius jau analizuoja sekančią, siekiant sutaupyti ir efektyviai panaudoti procesoriaus darbo laiką.

Konkrečiu atveju komanda yra EB – operacijos kodas parodantis, kad tai vidinis artimas jmp, AA yra vieno baito poslinkis pridedamas prie IP reikšmės esančios komandos vykdymo metu, taip pat reikia išplėsti pagal ženklo plėtimo taisyklę AA => FFAA. Naują IP skaičiuojame taip 2659+FFAA=2603

	2659
+	FFAA
1	2603

Papildomas vienetas niekur nesaugomas, IP registrą sudaro tik 4 šešioliktainiai skaitmenys, perteklius niekur nenaudojamas.

Besąlyginis valdymo perdavimas

Besąlyginiui valdymo perdavimui priklauso komandos JMP, CALL, RET, INT, IRET. Jas atpažinęs procesorius vykdo netikrindamas jokių sąlygų.

Besąlyginis valdymo perdavimas gali būti:
Išorinis arba vidinis, tiesioginis arba netiesioginis.
Ką tai reiškia?
Vidinis – valdymo perdavimas vyksta segmento viduje, vadinasi bus keičiama tik IP reikšmė.
Išorinis – valdymo perdavimas vyksta visos atminties ribose – bus keičiama ir CS ir IP reikšmės.

Tiesioginis – poslinkis, nauja IP reikšmė arba naujos CS ir IP reikšmės imamos tiesiogiai iš vykdomo kodo (yra baituose po komandos operacijos kodo)

Netiesioginis – Naujos IP (arba CS ir IP) reikšmės randamos naudojant adresavimo baitą (reg dalis reiškia OPK kodo plėtinį, o ne kažkokį registrą). Operandas atmintyje parodo iš kur reikės pasiimti naujas registrų IP (arba CS ir IP) reikšmes. **Jei mod=11**, tai operandas yra žodinis registras, vadinasi IP registrui priskiriama to žodinio registro reikšmė, o ne einama į atmintį ieškoti reikšmės (mod=11 neleistinas išorinio netiesioginio atveju).

Artimas – tiesioginio tipas, kai valdymas perduodamas mažu atstumu [-128; 127] baitai, todėl poslinkis užrašomas viename baite. Pridedant prie IP reikšmės esančios komandos vykdymo metu poslinkis išplečiamas iki 2 baitų pagal ženklo plėtimo taisyklę.

Prieš valdymo perdavimo komandas galimi segmento keitimo prefiksai, tačiau jie pakeičia tik operando atmintyje segmentą.

JMP komanda: vykdomas besąlyginis valdymo perdavimas į nurodytą atminties vietą, visiškai neįsimenant grįžimo adreso.

JMP tipas	*Operacijos kodas*	*Atliekami veiksmai*
Vidinis artimas	*EB*	Po operacijos kodo imamas vieno baito poslinkis, kuris yra išplečiamas iki 2 baitų pagal ženklo plėtimo taisyklę ir pridedamas prie IP reikšmės esančios komandos vykdymo metu.
Vidinis tiesioginis	*E9*	1. Po operacijos kodo paimami 2 baitai poslinkio 2. Kadangi pirmas jaunesnysis, antras vyresnysis jie sukeičiami vietomis 3. Gautas skaičius pridedamas prie IP reikšmės esančios komandos vykdymo metu.
Išorinis tiesioginis	*EA*	1. Po operacijos kodo paimami 4 baitai. 2. Jie priskiriami CS ir IP registrams tokiu eiliškumu: • IP jaunesnysis • IP vyresnysis • CS jaunesnysis • CS vyresnysis
Vidinis netiesioginis	*FF plėtinys* 100*	1. Po operacijos kodo analizuojamas adresacijos baitas (kuris gali būti su poslinkiu, priklausomai nuo mod reikšmės). Reg dalis reiškia opk plėtinį, r/m dalis operandą „registras/atmintis“ 2. Jei operandas yra atmintyje, t.y. mod nelygu 11, tada nueinama į tą atminties vietą į kurią rodo operandas ir paimami 2 baitai (IP jaunesnysis, IP vyresnysis), jei mod=11, tada IP registrui priskiriama žodinio registro reikšmė (kokio nurodo r/m dalis)
Išorinis netiesioginis	*FF plėtinys* 101*	1. Po operacijos kodo analizuojamas adresacijos baitas (kuris gali būti su poslinkiu, priklausomai nuo mod reikšmės). Reg dalis reiškia opk plėtinį, r/m dalis reiškia operandą atmintyje. 2. Mod negali būti 11, todėl operandas yra atmintyje, nueinama į tą atminties vietą, į kurią rodo operandas ir paimami 4 baitai tokiu pat eiliškumu kaip ir išorinio tiesioginio atveju.

CALL komanda: vykdomas besąlyginis valdymo perdavimas į nurodytą atminties vietą steke įsimenant grįžimo adresą (IP arba CS ir IP reikšmes, priklausomai nuo to, kurie registrai yra keičiami).
Kadangi veiksmai yra atliekami tie patys, kaip ir tokio paties tipo JMP atveju, tik prieš atliekant juos į steką padedamos keičiamų registrų reikšmės (išorinio atveju PUSH CS, PUSH IP, vidinio atveju PUSH IP), todėl pateikiami tik tipai ir jų operacijos kodai

CALL tipas	*Operacijos kodas*
Vidinis tiesioginis	*E8*
Išorinis tiesioginis	*9A*
Vidinis netiesioginis	*FF plėtinys* 010*
Išorinis netiesioginis	*FF plėtinys* 011*

*OPK *plėtinys* saugomas adresacijos baito ***reg*** dalyje

RET komanda: vykdomas grįžimas iš procedūros, kuri buvo iškviesta naudojant komandą CALL. Grįžimo adresas imamas iš steko.

Jei RET komanda yra vykdomas **vidinis** grįžimas iš procedūros (paprogramės), tada iš steko paimama tik IP reikšmė (POP IP)
Jei RET komanda yra vykdomas **išorinis** grįžimas iš procedūros (paprogramės), tada iš steko paimamos ir CS ir IP reikšmės: POP IP, POP CS

Po reikšmių paėmimo iš steko, žiūrima ar reikalingas **steko išlyginimas**.
Jei taip, tada SP:=SP+bet.op.
Jei ne, SP reikšmė nekeičiama

Jei nėra steko išlyginimo RET komandos ilgis yra 1 baitas (tik opk).
Jei yra **steko išlyginimas** RET komandos ilgis yra 3 baitai (operacijos kodo 1 baitas ir 2 baitai betarpiškas operandas, kur pirmas baitas yra jaunesnysis antras vyresnysis todėl prieš pridedant prie SP reikės sukeisti vietomis)

Ret operacijos kodai:

RET tipas	**Su steko išlyginimu**	**Be steko išlyginimo**
Vidinis	*C2*	*C3*
Išorinis	*CA*	*CB*

INT komanda: vykdomas valdymo perdavimas į pertraukimo apdorojimo programą.

Komanda turi vienintelį formatą (neskaitant INTO komandos):
Pirmas baitas yra operacijos kodas CD
Antras baitas reiškia skaičių n (vykdoma komanda INT n), nustatomas pertraukimo komandos valdymo numeris.
Pastaba: INT 3 mašininiam kode gali būti saugoma CD 03 arba CC.

Komandos **INT n**
Atliekami veiksmai:

1. PUSH SF
2. PUSH CS
3. PUSH IP
4. IF=0; (IF flagas)
5. TF=0; (TF flagas)
6. Pradedant absoliučiu adresu 4*n imami 4 baitai:
   - IP jaunesnysis
   - IP vyresnysis
   - CS jaunesnysis
   - CS vyresnysis

(šie skaičiai priskiriami CS ir IP registrams).
INTO yrą sąlyginis atvejis, todėl aprašytas prie sąlyginio valdymo perdavimo.

IRET komanda: atlikus pertraukimo procedūroje numatytą darbą valdymas grąžinamas pertraukimą iškvietusiai programai.
Operacijos kodas ***CF***.

Atliekami veiksmai:

1. POP ***IP***
2. POP ***CS***
3. POP ***SF***

Sąlyginis valdymo perdavimas

Sąlyginis valdymo perdavimo komandos kitaip nei besąlyginio pirmiausia tikrina ar išpildyta kokia nors sąlyga, tada tik vykdo valdymo perdavimą. Jei sąlyga nebuvo patenkinta komanda neatlieka jokių veiksmų ir vykdomas toliau po komandos ir esantis kodas.

Sąlyginiam valdymo perdavimui priklauso 17 atvejų Jmp (su sąlyga) J raidė ir po jos tam tikra sąlyga. LOOP atvejai ir INTO komanda.

Visais atvejais išskyrus INTO komandą operacijos kodas užima vieną baitą, po jo eina vieno baito poslinkis, kurį prieš pridedant prie IP reikia išplėsti iki 2 baitų pagal ženklo plėtimo taisyklę. Kadangi komandos pavadinimas iškart leidžia suprasti koks tai sąlyginis valdymo perdavimas, tai operacijos kodų įsiminti nereikia, todėl jie atskirai nėra pateikti.

Jei viename langelyje surašyti keli komandos pavadinimai reiškia jie yra visiškai vienodi mašininio kodo ir komandos vykdymo požiūriu. Tuomet, kurį pavadnimą naudoti pasirenka pats programuotojas rašydamas programą asembleriu.

INTO komanda – vykdyti INT 4, jeigu flagas OF=1.
Operacijos kodas ***CE***.
Vykdant šią komandą tikrinama sąlyga yra: ar OF=1.
Jei OF=0, tada INT 4 komanda nevykdoma, vykdomas kodas esantis po šios komandos kodo segmente.
Jei OF=1, tada įvykdoma INT 4 komanda

Ciklų komandos (LOOPai):

Visi LOOP atvejai veikia kaip vidinis artimas JMP (turi vieno baito operacijos kodą ir vieno baito poslinkį)

Atliekami veiksmai:
1) Sumažinti CX registro reikšmę vienetu
2) Tikrinti sąlygą:
   - Jei sąlyga tenkinama, tada peršokti su vieno baito poslinkiu (vieno baito poslinkis išplėstas pagal ženklą pridedamas prie IP esančio komandos vykdymo metu)
   - Jei sąlyga netenkinama, tada eiti vykdyti sekančios komandos, į kurią komandos vykdymo metu rodė CS ir IP reikšmės

Komanda	***Peršokimui su vieno baito poslinkiu būtina sąlyga***
LOOP	gautas CX nelygus 0000
LOOPE LOOPZ	gautas CX nelygus 0000 **IR** flagas ZF=1
LOOPNE LOOPNZ	gautas CX nelygus 0000 **IR** flagas ZF=0

Pastaba1: Sąlyga tikrinama PO TO kai CX sumažinamas vienetu
Pastaba2: IP komandos vykdymo metu rodo į sekančią už LOOP komandą kodo segmente
Pastaba3: Net ir tada, kai antrąjame žingsnyje gauta neteisinga sąlyga šios komandos CX reikšmę vis tiek sumažina vienetu

Sąlyginiai jmp:

Vykdant sąlyginius JMP atvejus tikrinama nurodyta sąlyga pagal registrų CX ir SF reikšmes (atskirus SF požymius).

- **Jei sąlyga patenkinama** imamas vieno baito poslinkis po operacijos kodo, išplečiamas iki 2 baitų pagal ženklo plėtimo taisyklę ir pridedamas prie IP esančio komandos vykdymo metu.
- **Jei sąlyga nepatenkinama** neatliekami jokie veiksmai, einama vykdyti sekančios komandos.

Komandos pavadinimas	*Pavadinimo prasmė*	*Tikrinama sąlyga*
JO	Jump if Overflow	OF=1
JNO	Jump if Not Overflow	OF=0
JNAE JB JC	Jump if Not Above nor Equal Jump if Below Jump if Carry	CF=1
JAE JNB JNC	Jump if Above or Equal Jump if Not Below Jump if Not Carry	CF=0
JE JZ	Jump if Equal Jump if Zero	ZF=1
JNE JNZ	Jump if Not Equal Jump if Not Zero	ZF=0
JBE JNA	Jump if Below or Equal Jump if Not Above	CF=1 arba ZF=1 (bent vienas)
JA JNBE	Jump if Above Jump if Not Below nor Equal	CF=0 ir ZF=0 (reikalingi abu)
JS	Jump if Sign	SF=1
JNS	Jump if Not Sign	SF=0
JP JPE	Jump if Parity Jump if Parity Equal	PF=1
JNP JPO	Jump if Not Parity Jump if Parity Odd	PF=0
JL JNGE	Jump if Lower Jump if Not Greater nor Equal	SF nelygu OF
JGE JNL	Jump if Greater or Equal Jump if Not Lower	SF=OF
JLE JNG	Jump if Lower Or Equal Jump if Not Greater	ZF=1 arba (SF nelygu OF) (bent vienas)
JG JNLE	Jump if Greater Jump if Not Lower nor Equal	ZF=0 ir SF=OF (reikalingi abu)
JCXZ	Jump if CX Zero	CX=0

J raidė reiškia žodį Jump

N – not, nor. **O** – odd. Sąlygos paneigimas.

C,**Z**,**S**,**P** – reiškia atitinkamus flagus

A – above (skaičiuose **be** ženklo), **G** – greater (skaičiuose **su** ženklu) reiškia „daugiau“

B – below (skaičiuose **be** ženklo), **L** – lower (skaičiuose **su** ženklu) reiškia „mažiau“

E – equal – lygu

JO, JNO atveju O raidė reiškia OF flagą

Eilutinės komandos

Eilutinės komandos ir jų atliekami veiksmai

Yra penkios eilutinės komandos MOVS, SCAS, CMPS, LODS, STOS.

Programuojant naudojamos su paskutine raide B arba W t.y. turime 10 komandų ***MOVSB, MOVSW, SCASB, SCASW, CMPSB, CMPSW, LODSB, LODSW, STOSB, STOSW.***

B raidė reiškia, kad veiksmus eilutinė komanda atlieka su baitais

W raide reiškia, kad veiksmus eilutinė komanda atlieka su žodžiais (2baitų dydžio laukais).

Visos šios komandos naudojamos be operandų, jų operacijos kodai užima vieną baitą, kur paskutinis bitas yra w bitas:
0 – jei komandos vardas baigiasi raide **B**
1 – jei komandos vardas baigiasi raide **W**

Įvykdžius eilutinę komandą kaskart yra koreguojamos registrų SI arba/ir DI reikšmės. Tačiau, jei kažkuris iš registrų komandoje nebuvo panaudotas, tai jo reikšmė išlieka nepakitusi.

Naudojamų indeksinių registrų reikšmės keičiamos automatiškai po komandos įvykdymo, pagal tokias taisykles:
Reikšmė yra mažinama, jei flagas DF=1
Reikšmė yra didinama, jei flagas DF=0
Reikšmė keičiama per 2, jei komanda dirba su žodžiais (baigiasi raide W)
Reikšmė keičiama per 1, jei komanda dirba su baitais (baigiasi raide B).

Kitaip tariant, tai galima aprašyti ir tokia lentele:

Direction Flagas	*Paskutinė komandos pavadinimo raidė*	*„Delta“ (ką reiktų pridėti prie panaudoto indeksinio registro)*
0	B	+1
0	W	+2
1	B	-1
1	W	-2

Eilutinių komandų sąrašas:

Komandos pavadinimas	*Atliekamas veiksmas*	*Pavadinimo prasmė*
MOVSB	***mov*** es:[di], ds:[si]	Move String Byte/Word
MOVSW		
CMPSB	***cmp*** ds:[si], es:[di]	Compare String Byte/Word
CMPSW		
SCASB	***cmp*** AL, es:[di]	Scan String Byte
SCASW	***cmp*** AX, es:[di]	Scan String Word
LODSB	***mov*** AL, ds:[si]	Load String Byte
LODSW	***mov*** AX, ds:[si]	Load String Word
STOSB	***mov*** es:[di], AL	Store String Byte
STOSW	***mov*** es:[di], AX	Store String Word

Komandas, kuriomis atliekamas mov veiksmas vadinkime duomenų persiuntimo, o komandas, kuriose atliekamas cmp, vadinkime palyginimo.

Status Flag registro reikšmę keičia tik *palyginimo eilutinės komandos*. Vykdant palyginimus pačių operandų reikšmės nekeičiamos.

Duomenų persiuntimo eilutinės komandos keičia atminties baitų arba AX/AL (akumuliatoriaus) reikšmę.
Tačiau SF registras jas vykdant lieka nepakitęs.

Eilutinių komandų vykdymas ir prefiksai

Eilutinė komanda yra vykdoma taip:

1. Atliekamas komandai nurodytas veiksmas
2. Pagal „delta“ pakeičiamos registrų SI ir DI reikšmės (abu, arba kažkuris iš jų, priklausomai, nuo to, kuris arba abu buvo panaudoti atliekant komandoje nurodytą veiksmą)

Jei naudojamas segmento keitimo prefiksas, jis pakeičia su registru SI naudojamą segmentą iš DS į bet kurį kitą. Jei vykdant eilutinę komandą įvyksta pertraukimas segmento keitimo prefiksas pamirštamas (segmentas vėl tampa DS).
Eilutinėse komandose dažniausiai naudojamas ir pakartojimo prefiksas REP (REPeate), turintis du variantus:
(vienam langelyje surašyti prefikso pavadinimai reiškia lygiai tą patį, nėra skirtumo kuris naudojamas)

Prefikso asemblerinė mnemonika	*Prefikso mašininis kodas*	*z bito reikšmė*
REP REPE REPZ	***F3***	***1***
REPNE REPNZ	***F2***	***0***

Abu atvejai dvejetainėje sistemoje užrašomi tokiu baitu: ***1111 001z***.

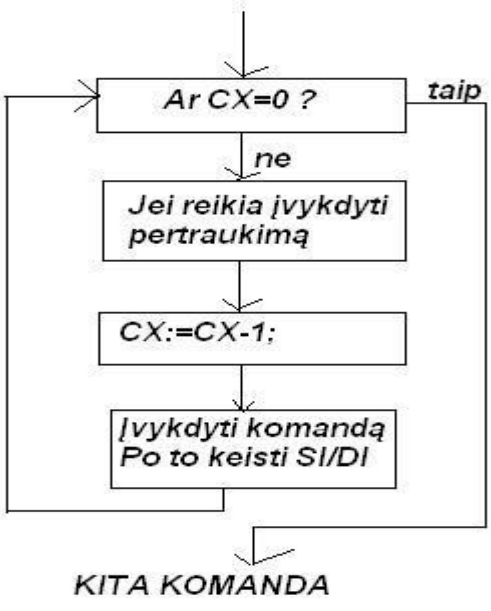
<u>Eilutinės komandos SU pakartojimu prefiksu vykdymas:</u>

1. Jei CX=0, visi šie veiksmai praleidžiami, ***einama vykdyti sekančią komandą***
2. Jei reikalinga įvykdyti pertraukimo procedūrą, ji įvykdoma
3. CX:=CX-1; t.y. sumažinti CX reikšmę vienetu
4. **Įvykdyti eilutinę komandą** (atlikti veiksmą, pakoreguoti indeksinius t. y. SI,DI registrus)
5. **Jei galioja abu šie punktai, reiškia einama vykdyti sekančią komandą**, jei bent vienas negalioja, pereiti į pirmą punktą
   - Vykdoma komanda yra CMPS arba SCAS
   - ZF flago reikšmė nelygi z bitui.

Eilutinės komandos su prefiksu vykdymą galima apibrėžti ir tokiomis schemomis.

PALYGINIMO
(SCAS, CMPS)

DUOMENŲ PERSIUNTIMO
(MOVS, STOS, LODS)



Mikroprogramavimo kalba (MPL)

Bendra informacija apie MPL

MPL yra mikroprogramavimo kalba, kuria užduodami elementarūs veiksmai procesoriui. Kiekviena asemblerinė komanda reiškia keletą MPL komandų. Ji yra žemiausio lygio programavimo kalba, dažniausiai vadinama asemblerinio lygio kalba.

MPL programos vykdomos mikroatmintyje, kurios dydis yra 256 vietos MPL komandų saugojimui, iš kurių kiekviena užima po 40bitų.

Naudojamos dvi mikrokomandos GATE ir TEST.

GATE reiškia procesoriui užduodamus veiksmus, kurie vykdomi netikrinant jokių sąlygų.

TEST reiškia tikrinimą, tikrinamas tam tikras registro bitas, ir jei jo reikšmė sutampa su nurodyta valdymas perduodamas tuo adresu.

MPL naudojami registrai

MPL naudoja tokius registrus:

Darbiniai (po 16 bitų):

A, B, C, D, X – visi užima po 16bitų

MBR – atminties buferinis – užima 16 bitų – per jį procesorius apsikeičia duomenimis su atmintim.

Būsenos valdymo:

MAR, PC, IR, SP – kiti registrai, kuriais nurodama adresas atmintyje, komandų skaitliukas (kaip, kad IP dirbant asembleriniam lygyje), steko viršūnė ir pan. Šie užima po 13bitų. Dirbant su jais galima suprasti, kad jie yra 16bitų registrai, kuriuose vyriausi 3 bitai visuomet nuliniai.

Visi registrai po 13 bitų išskyrus IR, kuris yra 16 bitų

Konstantiniai (po 16 bitų):

Šių registrų reikšmės niekuomet nekeičiamos, aparatūriškai nenumatyta galimybė juos pakeisti.

Jie saugo pastovias reikšmes:

Pavadinimas	*Šešioliktainė reikšmė*	*Dvejetainė reikšmė*	*Dešimtainė reikšmė*	
			Be ženklo	*Su ženklu*
1	0001	0000 0000 0000 0001	+1	+1
0	0000	0000 0000 0000 0000	0	0
-1	FFFF	1111 1111 1111 1111	+65535	-1
SIGN	8000	1000 0000 0000 0000	+32768	-32768
15	000F	0000 0000 0000 1111	+15	+15

TEST komandose naudojamas ir „fiktyvus“ registras žymimas Ø, visi jo bitai yra nuliniai.
Jis skirtas besąlyginiam valdymo perdavimui užrašyti.
GOTO VIETA;

Visų registrų išskyrus konstantinius reikšmės prieš vykdant komandas yra neapibrėžtos (nežinomos).

Mikrokomanda TEST

Tai yra valdymo perdavimo komanda, kuri MPL kalboje užrašoma tokioj formoj
IF BIT(k, reg)=b THEN GOTO VIETA;
k – kuris bitas tikrinamas, nuo 0 iki 15 (pats jauniausias, dešiniausias yra nulinis)
reg – kurio registro bitas tikrinamas, vienas iš registrų: A, B, C, D, MBR, IR, X, Ø
b – bito reikšmė, kuriai turi būti lygus reg registro k-tasis bitas
vieta – žymė (label'as) arba mikrokomandos, į kurią einama numeris iš intervalo [0;255]

Sąlyga tikrinama taip:
Jei ***reg*** registro ***k***-tasis bitas yra lygus ***b*** bito reikšmei, toliau vykdoma komanda adresu vieta.
Kitu atveju, komanda neatlieka jokių veiksmų, einama vykdyti kitos iš eilės esančios mikrokomandos.

Besąlyginis valdymo perdavimas realizuojamas naudojant bet kurį įmanomą k, fiktyvų registrą Ø ir b reikšmę nulį ir norimą vietą.
Tai tiesiog užrašoma taip:
GOTO vieta;

Mikrokomanda GATE

Šia komanda realizuojami persiuntimai tarp registrų, sumavimas ir duomenų persiuntimas iš atminties į ją.

Viena mikrokomanda gali būti užrašomi keletas suderinamų veiksmų. Todėl ***viena eilutė reiškia vieną mikrokomandą***.

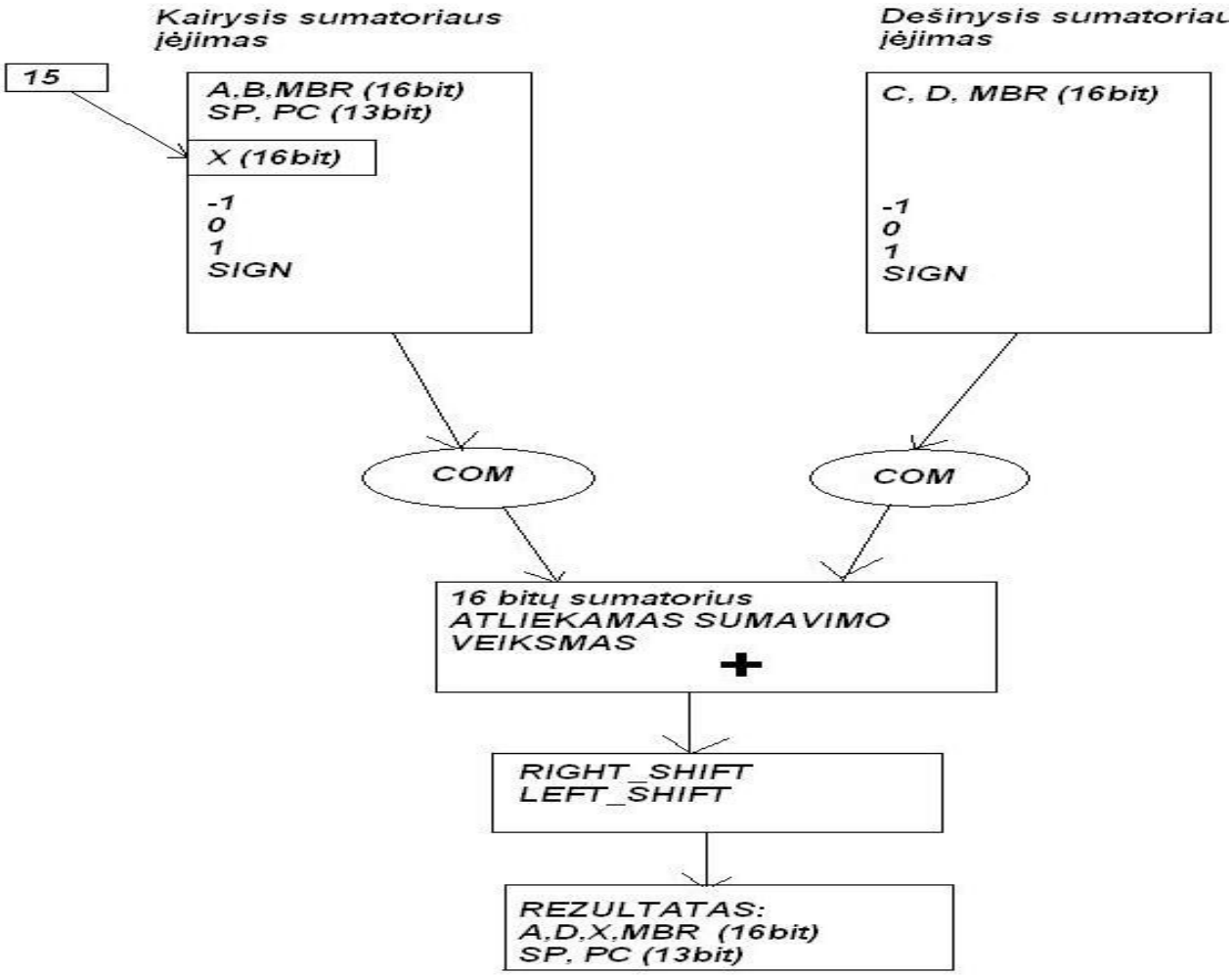
Siekiant išvengti neapibrėžtumų vykdant *vieną mikrokomandą įvyksta 3 atskiri pocikliai*:
1. persiuntimas tarp registrų, registrų reikšmių (galimai invertuotų) į sumatorių pasiuntimas
2. sumavimo veiksmas ir shiftai
3. sumatoriaus rezultato nusiuntimas į registrą/registrus ir duomenų apsikeitimas su atmintimi.

Reikšmių persiuntimai tarp registrų:
Jie realizuojami rašant kur=iš kur;
Tarkim reg1=reg2; registrui reg1 priskiriama reikšmė reg2

Galimi variantai:
Į ***MBR*** registrą galima pasiųsti: sumatoriaus rezultatą ir reikšmę iš atminties *MBR=MEMORY(MAR)*; - iš atminties žodis numeriu MAR siunčiamas į registrą MBR
Į ***MAR*** registrą galima pasiųsti: ***IR***, ***PC*** registrų reikšmes
Į ***PC*** registrą galima pasiųsti ***IR*** registro reikšmę
Į ***SP*** registrą galima pasiųsti tik rezultatą iš sumatoriaus

Sumatoriaus panaudojimas mikrokomandose

Sumatoriaus naudojimą galima apibrėžti tokia schema:



Komentarai apie sumatoriaus naudojimą:

- Sumatorius gali atlikti tik sudėties operaciją
- Naudojant sumatorių būtina į abu įėjimus (puses) pasiųsti po kurį nors vieną registrą.
- Į sumatoriaus įėjimus galima pasiųsti tik po vieną registrą
- Toje pačioje komandoje neleidžiama į sumatorių pasiųsti registro, kuris toje komandoje yra pakeičiamas naudojant paprastą priskyrimą. T.y.:
  Galima rašyti: *X=15; A=A+1;* nes į A siunčiamas sumavimo rezultatas, o ne atliekamas paprastas priskyrimas
  Negalima rašyti *X=15; MBR=X+1;* nes neapibrėžta, kokia X reikšmė pateks į sumatorių (15, buvusi prieš tai ar dar kas nors)
- Persiuntimai tarp kai kurių registrų gali būti įvykdomi per sumatorių pridedant nulį.
- Sumatorius komandoje gali būti naudojamas tik vieną kartą, bet rezultatas gali būti išsaugotas keliose vietose, tarkim į A ir D registrus 1 galima pasiųsti tokia mikrokomanda:
  A=0+1; D=0+1;
- ***Schemoje parodyti dalykai:***
  - Iš kokių registrų reikia rinktis po vieną nurodyta atskirose sumatoriaus pusėse
  - COM operaciją galima atlikti tik su atskiromis sumatoriaus pusėmis (kaire, dešine, abiem arba nė viena, taikoma tik atskiriem dėmenims)
  - Sumavimo veiksmą parodo + ženklas, jis yra skirtukas, tarp sumatoriaus pusių
  - RIGHT_SHIFT ir LEFT_SHIFT vykdomi, tik po sumavimo.
  - Nurodyta, kokiuose registruose galima saugoti atlikto sumavimo rezultatą.

Pastaba: negalima toje pačioje komandoje naudoti X=15; ir jo siųsti į sumatorių, nes gaunami neapibrėžtumai.

Sumatoriaus panaudojimo užrašymo taisyklė:

Sumatoriaus atliekamas veiksmas turi būti užrašytas laikantis sintaksės *rezultato registras = kairės pusės registras + dešinės pusės registras;*

COM funkcija gali būti taikoma tik kiekvienam iš registrų (dėmenims) atskirai, pagal poreikį, pvz:

MBR=1+COM(-1);
MBR=COM(1)+COM(-1);
MBR=COM(1)+(-1);
MBR=1+(-1);
yra teisingos komandos.

RIGHT_SHIFT ir LEFT_SHIFT gali būti taikomi tik sumavimo rezultatui:

X=RIGHT_SHIFT(COM(1)+0);
A=LEFT_SHIFT(0+COM(0));
yra teisingos komandos.

COM ir shiftų prasmė:

COM naudojamas invertuoti registro bitams. Invertuojamas kiekvienas bitas atskirai (t.y. vienetai pakeičiami nuliais, nuliai vienetais)

Reikšmė su priešingu ženklu gaunama invertuojant ir pridedant vienetą:
COM(sk)+1=-sk; atėmė po 1 iš abiejų pusių gauname:
COM(sk)=-sk-1; šį atvejį reikia atsiminti.

Sumatoriaus darbui rezultatui:
RIGHT_SHIFT reiškia, kad kiekvienas bitas pastumiamas per vieną vietą į dešinę, kairėje esančio bito vietoje atsiranda 0. (kai RIGHT_SHIFT'inamas neigiamas skaičius jis stipriai pakeičia savo reikšmę, tarkim RIGHT_SHIFT atlikus su reikšme -1 gautume +32767)
LEFT_SHIFT reiškia, kad kiekvienas bitas pastumiamas per vieną vietą į kairę, dešinėje esančio bitų vietoje atsiranda 0.
Komandų pavyzdžiai:

MBR=LEFT_SHIFT(1+0);
MBR=RIGHT_SHIFT(1+COM(0));

Jei dirbame su skaičiais be ženklo iš intervalo [0; 65535], t.y. Sumatoriaus rezultatą rašome skaičiaus be ženklo formoje, tada galime RIGHT_SHIFT ir LEFT_SHIFT veiksmus suprasti taip:
LEFT_SHIFT t.y. (skaičiaus daugyba iš dviejų) mod 65536
RIGHT_SHIFT t.y. (skaičiaus dalybos iš dviejų liekana) mod 65536

Išvestinės konstantos:

Veiksmas	*Šešioliktainė reikšmė*	*Dvejetainė reikšmė*	*Dešimtainė reikšmė*	
			Be ženklo	*Su ženklu*
COM(0)	FFFF	1111 1111 1111 1111	+65535	-1
COM(1)	FFFE	1111 1111 1111 1110	+65534	-2
COM(-1)	0000	0000 0000 0000 0000	0	0
COM(SIGN)	7FFF	0111 1111 1111 1111	+32767	+32767
MBR+COM(MBR)	FFFF	1111 1111 1111 1111	+65535	-1

Atvejis ***MBR+COM(MBR)*** naudojamas tuomet, kai reikia gauti kokią nors reikšmę „nenaudojant konstantinių registrų“ vienąkart naudodami sumatorių šią reikšmę galime pastumti į dešinę, į kairę arba nestumti išvis.

MBR+COM(MBR) atveju taikoma savybė: sudėję bet kokią dvejetainę reikšmę su invertuota ta pačia reikšme, gauname skaičių sudaryta iš vien vienetinių bitų.

Egzamino užduočių pavyzdžiai su sprendimais

Bendra informacija apie pateiktus sprendimus

Kiekvienas išspręstas uždavinys yra priskirtas kažkuriai iš temų. Temos pavadinimas baigiasi laužtiniais skliaustais ir juose įrašytu skaičiumi. Tas skaičius reiškia kiekį, kiek tos temos uždavinių šiuo metu yra konspekte, gali būti, kad naujesnėse versijose jų rasite ir daugiau.

Sprendimai pateikti siekiant apimti kuo daugiau atvejų, ir dalyje jų yra aprašyti net ir tie veiksmai, kurių nereikia atlikti duotąjame uždavinyje jį sprendžiant, bet į ką reiktų atsižvelgti, kai uždavinys atrodo panašus, bet iš tikrųjų kažkuri jo sprendimo dalis kažkiek skirasi.

Kiekvieno uždavinio sprendimas sudalyti į tokias dalis: ***uždavinio sąlyga, detalus sprendimas***, ***atsakymas***.

Vaikiška aritmetika [1]

Uždavinio sąlyga: Užrašykite dešimtainę reikšmę -59 viename žodyje šešioliktaine sistema.

Sprendimas:

Reikšmė turės būt šešioliktainis žodis, vadinasi tarp 0000 ir FFFF.

Neigiamas skaičius, patenka į skaičiaus su ženklu intervalą, tai tereikia dviejuose baituose užsirašyti -59.

1 būdas:

-59 užrašyti nemokame, bet mokame +59. Kadangi... 32+16=48... Tai 32+16+8+2+1 ir yra ko mums reikia. 0000 0000 0011 1011 = 003B.

Bet mums reikia reikšmės su priešingu ženklu. Pasinaudojame tuo, kad ženklą baitams ir žodžiams (*jokiu būdu nedarykit šito su slankiu kableliu, ten ženklas keičiamas tiesiog pakeičiant ženklo bitą!!!*) galima pakeisti atlikus porą veiksmų:

- Invertavimas
- +1

Invertavę turime 1111 1111 1100 0100, pridėję 1: 1111 1111 1100 0101 (FFC5).

2 būdas:

Prieš tai parodytu būdu gauname, kad +59 yra 3B.

Atliekame žodžių atimtį: 0000 – 003B, kadangi skolintis galima kiek tik norima, nes kas yra už žodžio ribų mūsų nedomina, tai tėra 1|0000 – 003B = FFC5.

Vengiant skolinimosi galima buvo rašyti ir taip: -59_{10} = -1 - (+58) = FFFF – 003A = FFC5, atimant pagal atimtį stulpeliu.

Atsakymas: FFC5.

Adresavimas [2]

UŽDAVINYS NR. 1

Uždavinio sąlyga: Registrų reikšmės yra DS=FE21, SS=3456, CS=CC31, ES=E341, BP=329A, BX=3675, SI=FA45, DI=F122. Apskaičiuoti operando absoliutų adresą pagal adresavimo baitą 6E.

Po adresavimo baito seka baitai: 89 AB

Sprendimas:

Tikslas – operando absoliutaus adreso radimas, pagal adresavimo baitą.

Žinome adresavimo baitą 6E. Užsirašome šį skaičių dvejetainėje sistemoje: 01101110.

Iš to turime: mod=01, reg=101, r/m=110.

Pagal mod ir r/m nustatome, koks tai yra operandas atminty ir su kokiu poslinkiu (mod nelygu 11, tai operandas atmintyje, o ne registras, be to tai pasakyta ir sąlygoje).

Pagal mod ir r/m nustatome, kad:

r/m=110, kai mod=01 yra BP+poslinkis.

Kadangi mod=01, reiškia poslinkis yra vieno baito, ir jį reikės išplėsti iki dviejų baitų pagal ženklą.

Efektyvaus adreso BP+poslinkis skaičiavimas:

BP=329A, vieno baito poslinkis yra 89. Kadangi 89 vyriausias bitas yra 1, plėsdami pagal ženklą turime poslinkį FF89. Pridedame jį prie BP:

	329A
+	FF89
1	3223

Kadangi skaičiuojame operando efektyvų adresą todėl į papildomą vienetą dėmesio nekreipiame (efektyvų adresą sudaro 4 šešioliktainiai skaitmenys). EA=3223.

Kokį segmentą naudoti?

Jei būtų segmento keitimo prefiksas, segmentą nustatytume pagal jį.

Kadangi prefikso nėra, žiūrime ar kombinacijoje panaudotas BP registras?

Taip ---> Segmentas SS

Ne ---> Segmentas DS.

Šiuo atveju BP kombinacijoje (formuojant efektyvų adresą) buvo panaudotas (***BP***+poslinkis). Vadinasi segmentas SS.

Todėl absoliutų adresą skaičiuosime pagal formulę:

AA = SS*10h + EA. (jei gautas AA būtų daugiau nei 5 skaitmenys papildomus ignoruotume, jei skaitmenų būtų mažiau nei 5 priekyje prisirašytume nulius)

	34560
+	3223
	37783

Atsakymas: 37783

UŽDAVINYS NR. 2

Uždavinio sąlyga: Registrų reikšmės yra CS=70DF; DS=76E1; SS=7D1F; ES=12CE; BP=631D; BX=443D; DI=FBB2; SI=AF1D. Apskaičiuokite operando atmintyje absoliutų adresą pagal adresavimo baitą 2E. Po adresavimo baito seka baitai 23 3C.

Sprendimas:

Tikslas – rasti operando atmintyje absoliutų adresą

Adresavimo baitas yra 2E. Užsirašome jį dvejetainėje sistemoje turime 00101110.

Iš jo nusistatome mod, reg, r/m reikšmes.

Gavome, kad:

mod=00, reg=101, r/m=110

Pagal mod ir r/m iš adresavimo baito lentelės nustatome, kad tai yra atvejis: tiesioginis adresas.

Tiesioginis adresas suprantamas kaip „vienišas poslinkis“ prie kurio nepridėti jokie registrai.

Ir tas poslinkis yra visuomet dviejų baitų – tiesioginis adresas iškart nurodo, koks yra operando efektyvus adresas.

Taigi sekantys du poslinkio baitai ir bus tiesioginis adresas, reikia atkreipti dėmesį, kad pirmas yra jaunesnysis, kitas vyresnysis, todėl šiuo atveju operando atmintyje efektyvus adresas yra EA=3C23.

Absoliutus adresas skaičiuojamas AA=seg.reg. * 10h + EA

Trūksta nustatyti koks segmento registras yra panaudotas absoliutaus adreso formavimui.

Pirmiausia žiūrime, ar yra prefiksas? Ne – prefikso nėra. Gali kilti pagunda sakyti, kad 2E reiškia cs: prefiksą, tačiau, 2E yra ne prefikso baitas, o adresacijos baitas.

Jei būtų prefiksas, jis nurodytų, koks yra operando atmintyje segmentas, tačiau šiuo atveju prefikso nėra, tai reiškia, kad operando atmintyje segmentas yra DS arba SS. Kuris iš jų?

Jei kombinacijoje panaudotas BP ---> SS

Jei kombinacijoje nepanaudotas BP ---> DS
Kadangi formuojant efektyvų adresą nepanaudotas BP, reiškia turime, kad segmentas yra DS.
Iš sąlygos DS=76E1, EA=3C23.
Skaičiuojame AA pagal formulę AA=DS*10h+EA.
Turime:

$$\begin{array}{rr} & 76E10 \\ + & 3C23 \\ \hline & 7AA33 \end{array}$$

Jei absoliutus adresas sudarytų daugiau nei 5 skaitmenys, į papildomus dėmesio nekreiptume, jei turėtume per mažai trūkstamų vietoje prisirašytume nulius. T.y. gavus 123456 AA būtų 23456. Turint 123 AA būtų 00123.

Atsakymas: 7AA33

Status Flag registras [1]

UŽDAVINYS NR. 1

Uždavinio sąlyga: Vykdydami baitų atimties operacija iš dešimtainio skaičiaus 160 atimame dešimtaini skaičių -28. Pradinis SF=7964. Kokia bus nauja SF registro reikšmė?

Sprendimas:
Tikslas – rasti naują SF reikšmę.
Sudarome tokią lentelę:
Pabrauktieji bitai nesikeis vykdant aritmetinę operaciją. (nenaudojamus žymiu žvaigždutėm)

Bitų reikšmės	********	**ODIT**	**SZ*A**	***P*C**
Pradinis SF	**0111**	**1001**	**0110**	**0100**
Naujas SF				

Atliekame sąlygoje nurodytą atimtį:
Pasiverčiame duotus skaičius į skaičius be ženklo, ir užrašome šešioliktainėje sistemoje (galima ir dvejetainėje)

$$\begin{array}{rr} & A0 \\ - & E4 \\ \hline 1 & BC \end{array}$$

JEI ATIMTĮ KEIČIATE SUDĖTIMI, DALIS FLAGŲ (AF, CF gali būti gauti neteisingi)

Dėl įvykusio pasiskolinimo iš už rezultato lauko ribų (fiksuojama vienetu, CF=1)
Užsirašome rezultato lauką dvejetainėje sistemoje, turime ***BC***h=*1011 1100*.
Vienetinių bitų kiekis yra 5, skaičius 5 yra nelyginis, todėl PF=0.
Taip pat, vyriausias bitas (pabrauktasis) yra ženklo bitas, todėl SF=1.
Jauniausias pusbaitis atlieka pasiskolinimą iš vyresnio? Taip, 0-4 reikia skolintis, vadinasi AF=1.
Ar rezultato laukas yra sudarytas tik iš nulinių bitų? Ne, ZF=0.

Beliko nustatyti naują OF reikšmę.
Į skaičius, kuriais operuojame ir rezultatą žiūrime kaip į dešimtainius skaičius su ženklu. Tuomet, žiūrime ar atlikę veiksmą gauname teisingą rezultatą, jei ne, tada OF=1, jei rezultatas teisingas OF=0.
Vienam baite reikšmių aibės dydis yra 256, skaičiaus su ženklu intervalas ***[-128; 127]***.
160 nepatenka į skaičiaus su ženklu intervalą, todėl pasiverčiame į skaičių su ženklu:
160-256=-96.
-28 patenka į intervalą, tai yra tinkamas skaičius su ženklu.
Gautas rezultatas: Bch=11*16+12=176+12=188 (nepatenka į intervalą).
Pasiverčiame į skaičių su ženklu 188-256=-68.
Ką tai reiškia, kad atlikome veiksmą
Iš -96 atimame -28 gauname -68
T.y.

-96-(-28)=-96+28=-68. Rezultatas tinkamas, perpildymo nėra, OF=0.

Papildome prieš tai naudotą lentelę:

Bitų reikšmės	********	**ODIT**	**SZ*A**	***P*C**
Naujas SF	**0111**	**1001**	**0110**	**0100**
Naujas SF	0111	0001	1011	0001
Naujas SF šešioliktainėj	7	1	B	1

Gautas atsakymas ir bus 71B1.

Atsakymas: 71B1

Supakuoti ir išpakuoti skaičiai [1]

Uždavinio sąlyga:
Duota registro reikšmė AX=ABFE, flagas AF=1 kokia bus AX reikšmė įvykdžius komandą AAA?
Sprendimas:
Tikslas – rasti naują AX reikšmę.
Žinome, kad komanda yra AAA. t.y. Išpakuotų skaičių sudėties korekcijos komanda.
Flagas AF=1
AX registro reikšmę išsirašome per du baito dydžio registrus AH ir AL
AH*=AB*
AL*=FE*
„bukai taikome AAA komandos algoritmą“ - ***AH ir AL šiuo atveju atskiri baito dydžio registrai***, nereikia plakti į vieną AX. Tai padarome tik gale užrašydami atsakymą. Vykdant AAA jie koreguojami atskirai.
Pritaikome algoritmą.

AAA komandos veikimo algoritmas	Algoritmo taikymas konkrečiu atveju
if*((AL and 0Fh) >9) or (AF=1)) **then*** *AL := **AL + 6*** *AH:= **AH + 1*** *AF:= 1* *CF:= 1* ***else*** *AF:=0* *CF:=0* ***endif*** *AL:=AL and 0Fh*	If(FE and 0Fh) >9 or (AF=1) then t.y.: if(E>9) or (AF=1) then... E>9 tai antrą sąlyga neberūpi, vis tiek sąlyga teisinga todėl vykdysim žaliai pažymėtąją dalį. Geltonoji dalis vykdoma bet kokiu atveju. Taigi atliekam žalio fono veiksmus: AL=FE+6=1***04, kadangi AL yra vieno baito perteklius neskaitomas.*** AH:=AB+1=AC AF=1 CF=1 *AL=04 and 0F=>04 (nunulinamas vyresnysis pusbaitis)* Turim AH=AC, AL=04, flagų niekas neklausė sąlygoj... taip, kad: AX=AC04

Atsakymas: AC04

Slankaus kablelio skaičiai (float'ai) [1]

Uždavinio sąlyga: Užrašykite dešimtainį skaičių -6,725 slankaus kablelio koprocesoriaus vidiniu formatu šešioliktaine sistema.

Sprendimas:

Tikslas - Užrašyti turimą dešimtainį slankaus kablelio skaičių nurodytu formatu.
Pasakyta, kad tai vidinis formatas. Vadinasi reikės užrašyti duotą skaičių dešimtyje baitų.
Šiame formate turime:
1 ženklo bitas
15 bitų charakteristikos (eilė+3FFFh)
1 i bitas
Mantisė iki pat galo (kol susidarys 10 baitų).

Turime dešimtainį skaičių -6,725
Skaičius neigiamas, todėl ženklo bitas bus 1

Ženklą užfiksavome, dabar galime elgtis kaip su teigiamu skaičiumi 6,725
Užsirašome turimą skaičių dvejetainėje sistemoje:
Sveikoji dalis 6 ir trupmeninė 725.

Veiksmai su sveikąja dalimi:
$6=\mathbf{110}_2$
Veiksmai su trupmenine dalimi:
Išsirašysime tokio ilgio seką, kokios užteks mantisei užpildyti.
0,725 * 16 = 11,6 ($11_{10}=\mathbf{1011}_2$)
0,6 * 16 = 9,6 ($9_{10}=\mathbf{1001}_2$)
0,6 * 16 = 9,6 ($9_{10}=1001_2$)
Pastebime ciklišką (pabrauktos dalys kiek bedaugintume kartosis), turimas skaičius (+6,725) dvejetainėje bus:
110,**1011**(**1001**)
Skliaustuose nurodytą dalį kartosime tiek kartų kiek prireiks mantisei užpildyti.
Sudarome normalizuotą formą 1,mantisė.
Bendroji forma yra i,mantisė (normalizuotos atveju i=1)
Turime:

$-1{,}10\ 1011\ (1001) * 2^{2}$

Eilė=**2**
Charakteristika: eilė+3FFFh = 4001h
Mantisė: 10 1011 (1001) skliaustelių turinį kartosime tiek kartų kiek prireiks.

Užrašome tai ką turime dvejetaine sistema, ir po verčiame į šešioliktainę:

Turime:
ženklo bitas 1, charakteristika 4001h = 100 0000 0000 0001, i bitas 1
Mantisė 10 1011 (1001)

Pirmi penki baitai:

1100	*0000*	*0000*	*0001*	*1101*	*0111*	*0011*	*0011*	*0011*	*0011*
C	***0***	***0***	***1***	***D***	***7***	***3***	***3***	***3***	***3***

Kiti penki baitai:

0011	*0011*	*0011*	*0011*	*0011*	*0011*	*0011*	*0011*	*0011*	*0011*
3	***3***	***3***	***3***	***3***	***3***	***3***	***3***	***3***	***3***

Atsakymas: C0 01 D7 33 33 33 33 33 33 33

Valdymo perdavimas [4]

Besąlyginis valdymo perdavimas [3]

UŽDAVINYS NR.1

Uždavinio sąlyga:

Registras SS=ABCD, registras SP=0002, registras BP=AF00, registras CX=0010. Kokia bus steko segmento baito su adresu FFFE reikšmė šešioliktainėje sistemoje, įvykdžius komandą:

1234 9A EBFE6789 call text (1234 yra poslinkis kodo segmente)

Sprendimas:

Uždavinio tikslas – nustatyti steko segmente esančio nurodyto konkretaus baito reikšmę šešioliktainėje sistemoje.

Pirmiausia, įvardinta, kad vykdoma komanda call, ja vykdomas valdymo perdavimas, vadinasi bus keičiama IP reikšmė, arba CS ir IP reikšmės.

Iš operacijos kodo 9A nustatome, koks tai call komandos tipas – paaiškėja, kad išorinis tiesioginis.

Išorinis reiškia, keisim CS ir IP reikšmes. Tiesioginis reiškia, kad naujos IP ir CS reikšmės eina iškart po operacijos kodo.

Tačiau, vėlgi, koks uždavinio tikslas? Domina konkretaus baito steke reikšmė.

CALL komanda į steką įrašo CS ir IP, arba tik IP reikšmę priklausomai nuo tipo, t.y. jei CALL tipas yra toks, kad keičiamas tik IP registras, tai į steką bus įrašyta tik jo reikšmė. Jei keistųsi ir CS ir IP, tai į steką pasidėtume: PUSH cs, PUSH ip.

Šiuo atveju valdymo perdavimas išorinis, vadinasi keičiamos CS ir IP reikšmės, ir abi jos bus įrašomos į steką veiksmais PUSH cs, PUSH ip.

PUSH veiksmas vykdomas taip:

1) SP:=SP-2; (kituose žingsniuose naudojamas gautasis SP)
2) SS:SP įrašomas jaunesnysis baitas
3) SS:(SP+1) įrašomas vyresnysis baitas

Į steką reikės dėti cs ir ip reikšmes. CS iš uždavinio nustatyti neįmanoma, o ip yra IP komandos vykdymo metu.

IP registras komandos vykdymo metu rodo į sekančios vykdomos komandos pradžią, šiuo atveju vykdomos komandos pradžia yra 1234, o sekančios komandos sužinotume pridėję dabar vykdomos (visos call) komandos ilgį baitais.

Šiuo atveju operacijos kodas 9A. Tai išorinis tiesioginis formatas, iš formato nusistatome, kad po operacijos kodo eina dar 4 baitai, reiškia komandos ilgis yra 5 baitai. Pridedame: 1234+5=1239h

Dedame reikšmes į steką:

CS nežinome, IP=1234, tai pasižymim, komandos vykdymo metu: CS=????, IP=1239.

Pradinis SP=0002, push cs, push ip veiksmus pavaizduojame tokia lentele:

Atkreipiu dėmesį, kad lentelė sudaryta SP mažėjimo tvarka

SP	*Baito reikšmė (hex)*	*Prasmė*
0002	nežinoma	Šio baito reikšmė nesikeičia
0001	??	CS vyresnysis
0000	??	CS jaunesnysis
FFFF	12	IP vyresnysis
FFFE	**39**	**IP jaunesnysis**

Radome reikiamą reikšmę uždavinys išspręstas. Eilutėje duomenys paryškinti, reikalingo baito reikšmė pabraukta.

Atsakymas: 39h (h raidė nebūtina, vis tiek sąlygoj pasakyta, kad atsakymą reikia pateikti šešioliktainėj sistemoj)

UŽDAVINYS NR. 2

Uždavinio sąlyga: Registrų reikšmės yra: DS = 21FE, SS = 5634, CS = 0ADF, ES = 41E3, BP = 9A32, BX = 7100, SI = 0011, DI = 22F1.
Apskaičiuoti procedūros tolimo iškvietimo absoliutų adresą:
71EA 2E FF 98 D9 00 call cs : number (71EA – poslinkis kodo segmente)

Sprendimas:

Tikslas – rasti procedūros iškvietimo absoliutų adresą.

Sąlygoje pasakyta, kad komanda yra call.
Analizuojame pateiktą mašininio kodo fragmentą:
2E – segmento ketimo prefiksas (segmentas CS), kuris bus panaudotas operandui atmintyje
FF – iš to sužinome, kad call yra netiesioginis
98 – adresacijos baitas 98h=1001 1000
mod=10, reg=011, r/m=000 (reg yra OPK išplėtimas, parodantis, kad tai yra IŠORINIS call).
mod=10 ir r/m=000 reiškia, kad operando efektyvus adresas formuojamas taip: BX+SI+2 baitų poslinkis.
2 baitų poslinkis eina po adresacijos baito.
D9 – jaunesnysis poslinkio baitas
00 – vyresnysis poslinkio baitas.
Efektyvus adresas formuojamas taip BX+SI+00D9.
Skaičiuojame BX+SI:

BX+SI:		***BX+SI+poslinkis:***
+ 7100	>>>>>	+ 7111
0011		00D9
7111		71EA

Reiškia reikia iš atminties vietos *CS:71EA*. Reiktų paimti Naujas CS ir IP reikšmes.
CS:71EA yra duotasis kodo segmento gabaliukas.
Imame keturis baitus: 2E FF 98 D9.
2E – IP jaunesnysis
FF – IP vyresnysis
98 – CS jaunesnysis
D9 – CS vyresnysis.
Turime procedūros iškvietimo CS ir IP reikšmes:
CS=D998, IP=FF2E.
AA=CS*10h+IP

Skaičiuojame AA.

```
   D9980
+  FF2E
--------
  E98AE
```

Atsakymas: E98AE

UŽDAVINYS NR. 3

Uždavinio sąlyga: Registrų reikšmės yra: DS=FE21, SS=3456, CS=C131, ES=3EE3, BP=92A2, BX=C5D6, SI=45FA, DI=22F1, SP=FFE4. Kokia bus registro SP reikšmė įvykdžius grįžimo iš procedūros segmento viduje komandą?
C2 08 00

Sprendimas:
Tikslas – rasti SP reikšmę po duotos komandos įvykdymo.
Pasakyta, kad komanda vykdomas grįžimas iš procedūros, vadinasi tai yra komanda RET. Jei būtų pasakyta, kad grįžimas iš pertraukimo procedūros tai turėtume komandą IRET, šiuo atveju tai yra komanda RET.
Jos operacijos kodas yra vieno baito dydžio, šiuo atveju C2.

Mus domina, kaip keisis SP įvykdžius šią komandą.
Sąlygoje pasakyta „segmento viduje“ (arba galima suprasti iš operacijos kodo), kad vykdomas vidinis grįžimas iš procedūros, t.y. iš steko paimsime tik IP reikšmę. Tai reiškia veiksmą POP ip.
Taigi po to SP=SP+2.
(jei būtų išorinis, turėtume SP=SP+2+2=SP+4, nes vykdomi veiksmai POP ip, POP cs)
FFE4+2 = FFE6.

Kitas ***RET*** komandos veiksmas – ar vykdomas steko išlyginimas?
Jei taip, vadinasi sekantys du baitai po operacijos baito yra betarpiško operando jaunesnysis ir vyresnysis baitai.
Jei ne, SP papildomai nesikeis.
Pagal operacijos kodą C2 matome, kad steko išlyginimas vykdomas.
Betarpiškas operandas yra: 00 08.
Pridedame prie turimo SP:

$$\begin{array}{rr} & \text{FFE6} \\ + & \text{0008} \\ \hline & \text{FFEE} \end{array}$$

Pastaba: Jei atliktumėte veiksmus kitaip 0008+2, tai gali kilti pagunda rašyti 0010, bet tai yra šešioliktainiai skaičiai 8+2=A, todėl prie sąlygoje pridėto SP pridėtume 000A.
Be to, jei sudėję gautume daugiau nei 4 skaitmenis, juos turėtume ignoruoti (efketyviuose adresuose FFFF+1=0000)

Atsakymas: FFEE

Sąlyginis valdymo perdavimas (sąlyginiai jmp ir LOOP'ai) [2]

UŽDAVINYS NR. 1

Uždavinio sąlyga:

Įvykdžius nurodytą komandą apskaičiuoti sekančios vykdomos komandos absoliutų adresą, kai DS=21FE, SS=5634, CS=0ADF, ES=41E3, BP=9A32, SI=FFF1, DI=22F1, AX=0003, BX=0002, CX=0001, DX=0000, SF=0000.

4EF3 77 90 90 jnbe number (4EF3 yra poslinkis kodo segmente)

Sprendimas:

Uždavinio tikslas – apskaičiuoti sekančios vykdomos komandos absoliutų adresą. Vykdomos komandos absoliutus adresas išreiškiamas registrų pora CS:IP.

Sąlygoje paminėta mnemonika „jnbe“ reiškia sąlyginį valdymo perdavimą „jump if not below nor equal“. Sąlyginis valdymo perdavimas (LOOP'ai ir sąlyginai jmp) turi vienintelį formatą.

Pirmas baitas yra OPK, kitas baitas yra 1b poslinkis.

Turimas poslinkis išplečiamas pagal ženklą iki dviejų baitų ir pridedamas prie IP esančio komandos vykdymo metu.

Bet tik tada, jei sąlyginio jmp sąlyga patenkinama.

Jnbe reiškia „jump if not below nor equal“, arba kitaip „peršokti, jei nemažiau ir nelygu“, arba dar kitaip tariant „peršokti, jei daugiau“. Iš to reikia nustatyti, kad tikrins sąlygą flagai CF=0 IR ZF=0.

SF registrą išsiskleidžiame į dvejetainę sistemą

0000 0000 0000 0000

Matome, kad ZF ir CF flagai yra nuliai, vadinasi valdymo perdavimas bus vykdomas.

Jei valdymo perdavimas nebūtų vykdomas, tada sekančios vykdomos komandos absoliutų reikštų CS:IP, kur CS duotas pradinis, o IP yra IP komandos vykdymo metu t.y. CS:4EF3+2 (2 komandos ilgis) CS:4EF5.

Kadangi valdymo perdavimas vykdomas, skaičiuojame naują IP reikšmę taip:

IP komandos vykdymo metu rodo į sekančios komandos pradžią, todėl IP=4EF3+2=4EF5;

Prie IP esančio komandos vykdymo metu pridedamas poslinkis 90 išplėstas iki 2 baitų pagal ženklo plėtimo taisyklę FF90. Naujas IP bus 4EF5+FF90

```
   4EF5
+
   FF90
-------
1  4E85
```

Nauja IP reikšmė yra 4E85. IP sudaro keturi šešioliktainiai skaitmenys, tai į papildomą 1 dėmesio nekreipiame.

CS nesikeitė, todėl imame sąlygoj duotą jo reikšmę ir skaičiuojame absoliutų adresą pagal formulę CS*10h+IP.

```
     0ADF0
+
     4E85
---------
     0FC75
```

Atsakymas: 0FC75

UŽDAVINYS NR. 2

Uždavinio sąlyga:

Įvykdžius nurodytą komandą, apskaičiuoti registrų reikšmių sumą:
AL+BL+CL+DL+IP, kai AL=03, BL=02, CL=00, DL=01, AH=00, BH=01,
CH=02, DH=03, ES=0000, CS=ABCD, SS=1234, DS=FE21, SP=2222, SF=0000:
0100 E2 90 90 loop ... (0100 yra poslinkis kodo segmente)

Sprendimas:

Reikia rasti AL+BL+CL+DL+IP sumą.

AL,BL,DL nesikeitė vykdant LOOP komandą.

IP komandos vykdymo metu: 0100 + 2 0102 rodo į sekančią komandą, loop komandos ilgis yra 2 baitai:

- pirmas baitas OPK (E2)
- antras baitas yra vieno baito poslinkis

LOOP vykdomas taip:

- Sumažinti CX vienetu
- Ar CX dabar 0?
  - Taip → einama vykdyti komandos į kurią rodo komandos CS:IP komandos vykdymo metu
  - Ne → prie komandos vykdymo metu esančio IP pridedamas vieno baito dydžio poslinkis (nurodytas po operacijos kodo einančiame baite, mūsų sąlygos atveju sekantis po E2), kuris prieš pridedant yra išplečiamas pagal ženklą. Toliau vykdoma komanda, į kurią rodo naujai gautas CS:IP.

Vykdome LOOP:

- CX=0200 – 1 = 01FF (naujas CX=01FF, t. y. CH=01, CL=FF)
- CX nėra 0000 (yra 01FF), todėl: IP=IP+FF90 (IP=0102+FF90=1***0092***)

Turime naujas reikšmes:
AL=03 (iš sąlygos)
BL=02 (iš sąlygos)
CL=FF
DL=01 (iš sąlygos)
IP=***0092***

Skaičiuojame sumą: 03+02+FF+01+0092=(3+2)+(FF+1)+92=5+100+92=197

Atsakymas: 197

Pertraukimai [3]

UŽDAVINYS NR.1

Uždavinio sąlyga:

Atminties baituose su adresais nuo 00000 iki 000FF yra užrašytos reikšmės nuo -128 iki 127. Koks bus komandos INT 33h pertraukimo apdorojimo programos absoliutus adresas?

Sprendimas:

Tikslas rasti absoliutų kitos vykdomos komandos adresą (kur eitume vykdyti INT 33h komandos). Reikia suprasti pertraukimų veikimą, ir vektorių lentelę.

Skirtingų pertraukimo variantų yra 256, visų jų adresai per CS ir IP yra saugomi absoliučiais adresais
pertraukimo numeris * 4.
Ten saugomi 4 baitai tokia tvarka:
IP jaun. baitas.
IP vyr. baitas
CS jaun. baitas
CS vyr. baitas
(baitai sudėlioti atvirkščiai negu būtų rašoma įprastai)
CS=1234h
IP=5678h būtų sudėta 78 56 34 12
Šiuo atveju pertraukimo numeris yra 33h, dešimtainėje sistemoje turim 3*16+3*1=48+3=51, tai yra
penkiasdešimt pirmas pertraukimas.
Duota, kad atmintyje baituose nuo 00000 iki 000FF t.y. Adresuose nuo 0 iki 255 sudėtos reikšmės nuo -128 iki 127.
T.y.:
0 baite yra -128
1 baite yra -127
2 baite yra -126
.....
255 baite yra 127
Pastebimas dėsningumas, kad didėjant baito numeriui vienetu didėja ir pats skaičius.
Skaičius yra 128 vienetais mažesnis už baito numerį. Todėl paprasčiausiai galime pasižymėti, kad n-tajame baite yra skaičius: N-128.

Pertraukimo numeris 33h=51_{10}
Kadangi CS:IP rinkinys kiekvienam pertraukimui užima po 4baitus,
nulinės pertraukimo procedūros adreso CS:IP reikšmės sudėtos pradedant 0-uoju baitu (4*0)
pirmosios pertraukimo procedūros adreso CS:IP reikšmės sudėtos pradedant 4-uoju baitu (4*1)
n-tosios pertraukimo procedūros adreso CS:IP reikšmės sudėtos pradedant 4n-tuoju baitu
51-ojo pertraukimo procedūros adreso CS:IP reikšmės sudėtos pradedant nuo 51*4=204-tuoju baitu
Imsime keturių baitų reikšmes nuo 204tu eilės numeriu pažymėto atminties baito (pats pirmas yra 00000 t.y. nulinis).
Kokie skaičiai sudėti ten?
204tam 204-128=76, ***čia 76 yra dešimtainė reiškšmė***
toliau po vieną didėja
205tam 77
206tam 78
207tam 79
bet reikia persirašyt į šešioliktainus skaitmenis
76/16 = 4 ir liekana 12 (C raidė).
Turim 4C, 4D, 4E, 4F
IP reikšmė 4D4C

CS reikšmė 4F4E

skaičiuojam absoliutų adresą pagal formulę AA=CS*10h+IP

Daugindami iš 10h tiesiog prirašom dešinėj nulį (šešioliktainio daugyba iš šešiolikos, tą patį darom daugindami dešimtainį iš 10).

```
   4F4E0
+   4D4C
--------
   5422C
```

0+C=0+12=12=Ch (nepasiekta 16 riba, tai ir nieko nebus „mintyj“)
E+4=14+4=18=***1***2h (pasiekta 16 riba rašom 2, ***vienetas*** persikelia į kitą poziciją „mintyj“)
4+D+***1***=4+13+***1***=18=***1***2h (pasiekta 16 riba rašom 2, ***vienetas*** persikelia į kitą poziciją)
F+4+***1***=15+4+***1***=20=***1***4h (pasiekta 16 riba, rašom 4, ***vienetas*** „minty“)
4+***1***=5=5h

Alternatyvus sprendimas, kuris reikalauja tvirtesnio matematinio suvokimo:

Baituose:
00000: 80
00001: 81
00002: 82
…
33h + 33h = 66h
66h + 66h = CCh
Adresui didėjant po vieną ir reikšmė didėja po vieną.
000CC: ??

000CC: 80+CC=14C, įspraudžiam į vieną baitą turim 4C.
000CD: 4D
000CE: 4E
000CF: 4F

IP=4D4C
CS=4F4E

AA = 4F4E * 10h + 4D4C

Atsakymas: 5422C

UŽDAVINYS NR.2

Uždavinio sąlyga:

Kokia bus atminties baito su adresu 66323 reikšmė, įvykdžius programinio pertraukimo komandą INT 21h?
DS = FE21, SS = 5634, CS = C131, ES = 3EE3, SF = 04FF, BP = 92A2, BX = C5D6, SI = 45FA, DI = 22F1, SP = FFE4.

Sprendimas:

Tai yra vienas sunkesnių uždavinių, yra nestandartinis. Todėl sunku sugalvoti algoritmą. Tačiau visada sąlygoje duota viskas ko reikia atsakymui, tik reikia tai suprasti.

Klausiama baito atmintyje duotu adresu reikšmė.
Pasakyta, kad komanda yra INT 21h.
Vadinasi vykdomas valdymo perdavimas pertraukimo apdorojimo programai.
Ką ši komanda išvis daro su atmintim?

1. PUSH SF
2. PUSH CS
3. PUSH IP
4. flagai TF=0; IF=0

5. ima adresą iš vektorių lentelės, juo perduoda valdymą.

Pirmiausia matome, kad dirba su steku, stekas irgi yra ne kur kitur, o būtent atmintyje.
Duotas baito adresas 66323, reikėtų pagalvoti, o gal čia su steku viskas ir vyksta?
Ir iš tikrųjų, skaičiuojame steko viršūnės absoliutų adresą SS:SP.
SS=5634
SP=FFE4
SS:SP skaičiuosim taip:

$$\begin{array}{r} 56340 \\ + \\ \text{FFE4} \\ \hline 66324 \end{array}$$

Gavome reikšmę artimą reikalingam adresui. Dabar žiūrim, kaip į steką bus talpinami SF, CS ir IP registrai.

SF=04FF, CS=C131
IP reikšmės iš sąlygos nustatyti negalime, vadinasi jos ir neprireiks.

Taigi kas darosi su steku?

PUSH veiksmu mažiname SP reikšmę dviem, tada į tą vietą, kur rodo nauja SP reikšmė įdedam jaunesnįjį žodžio baitą, į SP+1 vyresnįjį.

Noriu atkreipt dėmesį, kad lentelė sudaryta adresų mažėjimo tvarka.

SS:SP (AA)	*SP (EA)*	*Baito reikšmė*	*Prasmė*
66324	FFE4	??	nežinom
66323	FFE3	**04**	SF vyresnysis
66322	FFE2	FF	SF jaunesnysis
66321	FFE1	C1	CS vyresnysis
66320	FFE0	31	CS jaunesnysis
6631F	FFDF	??	IP vyresnysis, nežinomas

Ta pati lentelė adresų didėjimo tvarka, kad būtų aiškiau:

SS:SP (AA)	*SP (EA)*	*Baito reikšmė*	*Prasmė*
6631F	FFDF	??	IP vyresnysis, nežinomas
66320	FFE0	31	CS jaunesnysis
66321	FFE1	C1	CS vyresnysis
66322	FFE2	FF	SF jaunesnysis
66323	FFE3	**04**	SF vyresnysis
66324	FFE4	??	nežinoma

Svarbu: šiuo atveju ir absoliutus ir efektyvus adresai atitinkamai keičiasi tokiu pat skirtumu, tačiau jei įvyktų perėjimas adresuose tarp FFFF ir 0000, tai vienu sumažėjus ar padidėjus EA turimas AA keistųsi ne taip, ir tektų pasiskaičiuoti naują AA reikšmę pagal formulę *AA=SS*10h+SP*

Vadinasi turim atsakymą pažymėtą geltonam fone baito reikšmę 04.

Atsakymas: 04

UŽDAVINYS NR. 3

Uždavinio sąlyga:

Atminties žodžiuose su adresais nuo 00000 iki 000FF yra užrašytos reikšmės nuo -1 iki -128. Koks bus komandos INT 13h pertraukimo apdorojimo programos absoliutus adresas?

Sprendimas:

Uždavinio tikslas – rasti naujas CS ir IP reikšmes įvykdžius komandą INT 13h.

Tai pertraukimo apdorojimo programos iškvietimo komanda, ji iš vektorių lentelės paima naujas CS ir IP reikšmes ir jomis perduoda valdymą.

Užsirašykime absoliučius adresus 00000 ir 000FF dešimtainėje sistemoje, žinome, kaip užpildyta seka nuo 0 iki 255 atminties baitų. Pirmieji 1024 atminties baitų (nuo 0 iki 1023 yra užpildyti vektorių lentelės duomenimis).

Kur yra naujos CS ir IP reikšmės? Pagal pertraukimų teoriją, jos imamos iš absoliutaus adreso 4*n, kur n yra pertraukimo numeris. Žinome, kad vykdoma komanda INT 13h, vadinasi pertraukimo komandos numeris yra 13h, kadangi patogiau dirbti dešimtainėje sistemoje, pasiverčiame 13h į dešimtainę skaičiavimo sistemą $13h=1*16^1+3*16^0=16+1=19_{10}$.

Turime n=19, apskaičiuojame 4*n=76. Vadinasi nuo 76 (skaičius dešimtainis!!!) baito keturiuose baituose yra naujos CS ir IP reikšmės. T.y. baituose su adresais 76, 77, 78, 79.

Tačiau kaip nustatyti kokie baitai ten yra?

Pasakyta, kaip užpildyti baitai su adresais nuo 0 iki 255. Jų tarpe yra ir 76,77,78,79 baitai, kurių reikšmės mus ir domina.

Taigi, dabar reikia tiesiog rasti tas reikšmes, jas radus pasiversti į šešioliktainę sistemą susirašyti į IP ir CS reikšmes bei apskaičiuoti absoliutų adresą CS:IP t.y. AA=CS*10h+IP. Tą ir padarome:

Žodžiai yra dviejų baitų dydžio laukai, todėl adresai keičiasi po 2.
Pabandome įžvelgti dėsningumą sudarydami lentelę:

Žodžio pradžios adresas (dešimtainėje sistemoje) **x**	Žodžio eilės numeris (dešimtainėje sistemoje) **y**	To žodžio reikšmė (dešimtainėje sistemoje) **z**
0	1 (pirmas)	-1
2	2 (antras)	-2
4	3 (trečias)	-3
6	4 (ketvirtas)	-4

Ką matome:

Pasižymėkime stulpelius iš kairės į dešinę kintamaisiais x, y, z.

Tuomet iš lentelės pastebime tokius sąryšius:

z=-y (z ir y yra priešingo ženklo)

(y-1)*2=x (vienu sumažinus y jis gaunamas dvigubai mažesnis už x)

Į antrąją lygybę įstatome -z vietoj y, turime:

(-z-1)*2=x.

Šiuo atvejo žodžių pradžios adresų reikšmės lyginės, o tarp mūsų reikalingų yra dvi lyginės reikšmės 76 ir 78, vadinasi reikia nustatyti žodžių reikšmes, kurie prasideda tais adresais.

Tai padarome taip:

Pasinaudojame sąryšiu (-z-1)*2=x. Turime x reikšmes 76 ir 78, jomis ir pasinaudodami randame z reikšmes:

Kai x=76	Kai x=78
(-z-1)*2=76 //:2 -z-1=38 //+1 -z=39 //*(-1) z=-39	(-z-1)*2=78 //:2 -z-1=39 //+1 -z=40 //*(-1) z=-40

Vadinasi žodis adresu 76 rodo į žodį, kurio reikšmė -39

Žodis adresu 78 rodo į žodį, kurio reikšmė -40.

Užrašome šiuos žodžius šešioliktainėje sistemoje. -39 telpa į vieno baito skaičių su ženklu intervalą, todėl

galime jį užrašyti kaip skaičių su ženklu viename baite šešioliktainėje sistemoje o žodį gauti atlikę išplėtimą pagal ženklą iš baito iki žodžio.
-39 pasiverčiame į skaičių be ženklo viename baite (pridėdami ar atimdami 256 kiek reikia kartų privedame iki intervalo [0;255]) -39+256=217. Pasivertę į šešioliktainę (galima ir per dvejetainę, kaip kam patogiau, bet vis tiek reikia gauti šešioliktainį skaičių) turime: D9.
Tą patį atliekame su -40. -40+256=216. Pasivertę į šešioliktainę turime D8.

Abu skaičius plečiame iki žodžio turime:
(adresai vis dar dešimtainiai!)
Adresu 76 žodį FFD9
Adresu 78 žodį FFD8.
Atmintyje įrašant žodžius pirmiau įrašomas jaunesnysis tada vyresnysi baitai, vadinasi turime:

Adresas (dešimtainėj)	*Baito reikšmė (šešioliktainėj)*	*Prasmė*
76	D9	IP jaunesnysis
77	FF	IP vyresnysis
78	D8	CS jaunesnysis
79	FF	CS vyresnysis

Gavome registrų reikšmes:
CS=FFD8, IP=FFD9.
Skaičiuojame absoliutų pertraukimo procedūros adresą, kuris reiškiamas registrū pora CS:IP.
AA=CS*10h+IP. Turime:

```
    FFD80
+
     FFD9
-------
1   0FD59
```

Kadangi absoliutų adresą sudaro penki skaitmenys į papildomą vienetą dėmesio nekreipiame.

<u>Alternatyvus sprendimas, bet reikalauja tvirtesnio matematinio suvokimo:</u>
Žodžiuose:
00000: FFFF
00002: FFFE
00004: FFFD

Arba baituose:
00000: FF
00001: FF
00002: FE
00003: FF
00004: FD
00005: FF
…
13h + 13h = 26h
26h + 26h = 4Ch

Adresui didėjant po 2 žodžio reikšmė mažėja po vieną.
0004C: ??
0004C: FFFF-4C/2=FFFF-26=FFD9 (IP)
0004E: FFD8 (CS)

Gauname FFFD8*10h+FFD9 = 0FD59

<u>Atsakymas:</u> 0FD59

Eilutinės komandos [2]

Kaip keičiasi registrų reikšmės vykdant eilutines komandas [1]

Uždavinio sąlyga:

Registrų reikšmės yra SI=EE8B ir DI=12CD, registras CX=0027, registras SF=FFFF. Kokia bus registrų SI ir DI reikšmių suma, įvykdžius komandą rep stosw?

Sprendimas:

Tikslas - Apskaičiuoti SI+DI įvykdžius eilutinę komandą, su pakartojimo prefiksu.

Komanda yra stosw, ji atlieka duomenų perkėlimą:

Komandos vardas baigiasi raide w, todėl ji atlieka veiksmus su žodžiais (jei baigtųsi raide b reiškia atliktų veiksmus su baitais)

Ši komanda adresu ***ES:DI*** įrašo žodinio registro *AX* reikšmę.

Kadangi komanda atliekamas duomenų persiuntimas (o ne palyginimai kaip CMPS arba SCAS) atveju, vadinasi komanda bus vykdoma CX kartų, t.y. iki tol, kol gausime CX=0000

Šiuo atveju CX=0027 vadinasi komanda bus vykdoma 27h kartų (pabrėžiu, kad skaičius 27 yra šešioliktainis.)

Kadangi komanda iš indeskinių (SI ir DI) naudoja tik DI registro reikšmę, vadinasi keičia tik ją.

Dirbama su žodžiais, todėl DI reikšmė keisis po 2, SI nepanaudotas, todėl įvykdžius komandą nepasikeis.

Kaip bus keičiamas DI? Didinamas po 2 ar mažinamas?

Tai nurodo Direction Flagas:

Išsirašome SF registro reikšmę, SF=FFFF:

1111 1**1**11 1111 1111.

Pabrauktasis bitas parodo, kad flagas DF=1, vadinasi DI reikšmė bus mažinamos.

Ką jau nustatėme:

Po šios komandos įvykdymo CX=0000, SI=EE8B (nesikeičia) ir reikia apskaičiuoti tik naują DI reikšmę, tada ir galėsime rasti SI ir DI sumą.

Naujas DI gaunamas 27h kartų po 2 mažinant DI reikšmę.

Todėl 27h padauginame iš dviejų ir gautą reikšmę atimame iš DI. Turėsim naują DI ir sudėję su SI turėsime atsakymą.

27h padauginimą iš dviejų patogu atlikti stulpeliu (nes dirbama su 2 baitų dydžio laukais), nes 27*2 galima per klaidą užrašyti 54, bet gautume neteisingą atsakymą todėl tiesiog sudedame stulpeliu:

```
  27
+ 27
----
  4E
```

Žinome per kiek sumažėjo DI reikšmė, randame naują DI reikšmę:

```
  12CD
—  4E
------
  127F
```

Turime SI=EE8B, DI=127F.

Sudedame (baitų/žodžių/registrų sumai apribojimų skaitmenų kiekiui nėra, nes tai nėra ribotas rezultato laukas, duomenys nesaugomi atmintyje):

```
   EE8B
+  127F
------
  1010A
```

Atsakymas: 1010A

Atminties būsenos pasikeitimas vykdant eilutinę komandą su pakartojimo prefiksu [1]

Uždavinio sąlyga: Registrų reikšmės SI = FFFF, DS = 1234, DI = FFFE, ES = 1234, registras CX = FFFF, registras SF = FF00. Duomenų segmento baito su adresu FFFF reikšmė yra 2. Kokia bus duomenų segmento visu baitų reikšmių suma įvykdžius komandą rep movsb?

Sprendimas:

Tikslas – rasti DS segmento visų baitų sumą įvykdžius eilutinę komandą su prefiksu.

Vykdoma movsb komanda atlieka duomenų persiuntimą.

Duomenys persiunčiami iš DS:SI į ES:DI.

Taip pat, įvykdžius komandą koreguojamos SI ir DI reikšmės.

Kadangi komanda baigiasi raide b reiškia atlieka veiksmus su baitais. Vadinasi vienakart įvykdžius komandą SI ir DI keisis po vieną. Didės ar mažės?

Iš SF registro imame DF požymį.

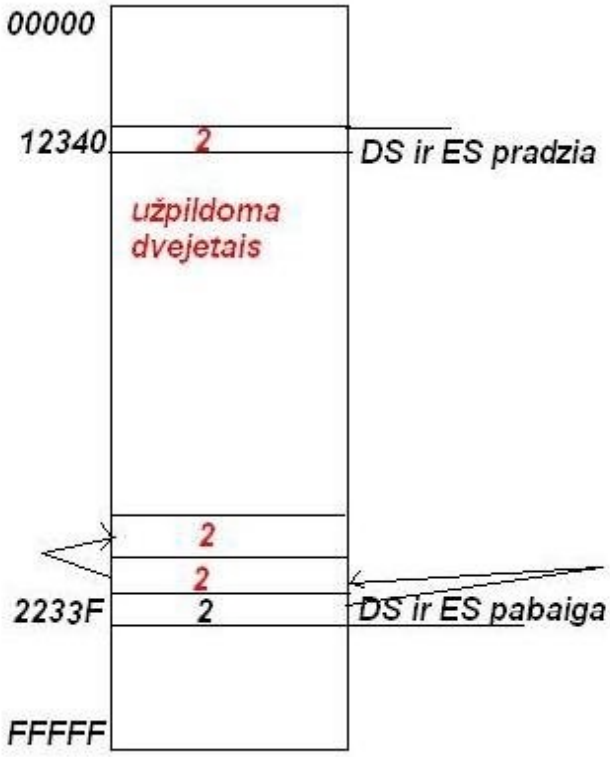
SF=FF00. Turime 1111 1***1***11 0000 0000.

Pabrauktasis rodo, kad DF=1.

Reiškia kaskart DI ir SI sumažės vienetu.

Kad geriau matytume, kas vyksta su atmintim nusipaišome visos atminties juostą. Atidedame galuose 00000 ir FFFFF (absoliutūs adresai).

Tada pasižymime DS:0000, DS:FFFF, ES:0000, ES:FFFF baitus, t.y. kur prasideda ir baigiasi kokie segmentai. Taip pat sekoje adresu DS:FFFF pasirašome, kad turime reikšmę 2, tai yra įvardinta sąlygoje.



CX=FFFF, reiškia komanda MOVSB vykdysime FFFF kartų.

Tiek kartų skaičiuoti sudėtinga, ieškome dėsningumo...

Įvykdome pirmą kartą:

Iš adreso DS:FFFF nukopijuojame baitą į ES:FFFE. Pagal brėžinį matome, kad ten pereina reikšmė 2.

Po įvykdymo sumažės SI ir DI po vieną, taip pat vienu sumažės CX.

Turėsime SI=FFFE, DI=FFFD, CX=FFFE.

Vykdydami daugiau kartų matome, kad vis kopijuojame reikšmę 2 per visą segmentą. Ir pasiekę ES segmento pradžią veiksmai baigiami, CX=0.

Turime visą DS segmentą užpildytą visi dvejetukais. Šiuo atveju jie visiškai sutampa ES ir DS segmentai.

Kas iš to?

Kadangi segmente nuo 0000 iki FFFF yra 10000h baitų, tai 10000h*2=20000h.

Pastaba: Šiuo atveju perėjimas tarp FFFF ir 0000 įvyko tik paskutinį kartą įvykdžius komandą ir nebeįdomus, tačiau jei taip nutinka, vykdant komandas į tai reikia atsižvelgti, kai pasiekus segmento galą peršokama į jo pradžią arba atvirkščiai (segmentai yra cikliniai)

Atsakymas: 20000h

MPL [3]
Kaip pasiųsti konkrečią skaičiaus reikšmę į registrus [2]

UŽDAVINYS NR. 1

Uždavinio sąlyga: Parašykite dvi mikrokomandas reikšmei -8 registruose A ir D nustatyti.
Sprendimas:
Iš sąlygos iškart matome, kad:

- Turės būti dvi mikrokomandos (t. y. lygiai dvi eilutės)
- Rezultatas turės atsirasti A ir D registruose
- Reikšmė yra lyginė, tikėtina, kad paskutinis veiksmas bus LEFT_SHIFT (atitinkantis daugybą iš dviejų, o koks skaičius dauginamas... ? -4, reiškia reikėtų pirmiausia pabandyti gauti -4).

-4 irgi yra lyginis, todėl galima mėginti irgi gauti LEFT_SHIFT pagalba iš... -2.

-2 galima gauti: (-1) + (-1) arba COM(1)+0 arba 0+COM(1) veiksmais.
Tai ir pamėginame parašyti, pasidėdami pradžiai reikšmę į MBR, nes šis registras leidžiamas abiejose sumatoriaus pusėse:
MBR=LEFT_SHIFT(COM(1)+0);
A=LEFT_SHIFT(MBR+0); D=LEFT_SHIFT(MBR+0);
(Tai yra vienas galimų teisingų atsakymų).

Pastaba: *A ir D reiškiniai privalo būti vienodi, nes kitaip tai reikštų negalimą veiksmą (sumatoriaus panaudojimą vienoje komandoje daugiau kaip vieną kartą).*

Gudriau mąstant buvo galima pastebėti, kad jau pirmoje komandoje iki LEFT_SHIFT'o buvo galima gauti -4 sudedant COM(1) su savimi:
A=LEFT_SHIFT(COM(1)+COM(1)); D=LEFT_SHIFT(COM(1)+COM(1));
Gavome tą patį per vieną mikrokomandą!

Bet... *Tai nebūtų teisingas šio uždavinio atsakymas*, nes uždavinyje prašoma reikšmę pasiųsti per **dvi** mikrokomandas, o čia panaudota viena.
Tuomet reikėtų rašyti:
A=LEFT_SHIFT(COM(1)+COM(1));
D=LEFT_SHIFT(COM(1)+COM(1));
(galimas teisingas ats.)

arba

A=LEFT_SHIFT(COM(1)+COM(1));
D=A+0;
(taip pat galimas teisingas ats.)

arba panašiai, svarbiausia prisiminti tinkamai sudėti skyrybos ženklus (laikytis sintaksės), prisiminti, kad A registras negalimas dešinėje, o D kairėje sumatoriaus pusėse, žinoti, kad atskiros mikrokomandos rašomos atskirose eilutėse ir pan.

Atsakymas: Daug variantų, keletą jų žr. sprendime.

UŽDAVINYS NR. 2

Uždavinio sąlyga: Parašykite vieną mikrokomandą, kuria dešimtainė reikšmė 32767 būtų užrašoma į MBR registrą. Nenaudokite konstantinių registrų.
Sprendimas:

Iš sąlygos pamatome, kad:

- Reikia gauti didžiulį skaičių. Prisiminę dvejetų laipsnius suprantame, kad $2^{15} = 32768$, vienu daugiau nei reikia, vadinasi mums reikia gauti $2^{15} – 1$, kas yra 8000h – 1 = 7FFFh
- Reikia vienos mikrokomandos, tai bus viena eilutė.
- Negalima naudoti konstantinių registrų: -1,0,1,SIGN,15 atkrenta.
- Rezultatas eis į MBR

Pasinaudosime tuo, kad COM(bet kas 16bitų) + tas pats bet kas 16bitų = FFFF. Iš FFFF (1111 1111 1111 1111) gauti 7FFF (0111 1111 1111 1111) galima tiesiog... Padarius RIGHT_SHIFT su FFFF (nes MPL naudoja ne aritmetinius ženklą besinešančius, o paprasčiausius loginius shift'us).

Kadangi FFFF gauti mokame, nes tam tereikia tą pačią reikšmę pasiųsti į abi sumatoriaus puses, vienoj pusėj panaudoti COM, kitoj ne.

Dar žinome, kad vienintelis registras einantis į abi sumatoriaus puses yra MBR, vadinasi FFFF gausime MBR + COM(MBR) arba COM(MBR) + MBR.

Belieka tik uždėti shift'ą, pasirūpinti skyryba ir... parašyti atsakymą.

Atsakymas:
Du galimi variantai:
MBR = RIGHT_SHIFT(MBR + COM(MBR));
arba
MBR = RIGHT_SHIFT(COM(MBR) + MBR));

Kaip patiems interpretuoti mikrokomandas [1]

Uždavinio sąlyga: Parašykite, kokia šešioliktainė reikšmė bus MBR registre įvykdžius šias mikrokomandas:
MBR = RIGHT_SHIFT(COM(-1) + 1); C=MBR;
A = LEFT_SHIFT(COM(MBR)+C); D=MBR; X=15;

Sprendimas:
Domina tik MBR reikšmė. Mąstome pocikliais:

Pirmoji mikrokomanda:
Pirmame kažkokia MBR reikšmė nueis į C, į sumatorių nueis COM(-1) t. y. 0000 ir 0001
Antrame 0000 ir 0001 bus sudėta, ir gausis 0001.
Trečiame 0001 bus pashiftinta per bitą į dešinę ir gausis 0000.

Tolesnėje mikrokomandoje į MBR niekas nebesiunčiama jau turime atsakymą, galime nebegaišti laiko tolesniam sprendimui.

Bet panalizuokime situaciją detaliau.

Šioje mikrokomandoje į C ateis nežinoma pradinė MBR reikšmė (po pirmo pociklio), o MBR taps 0001 (po antro pociklio).

Antroji mikrokomanda:
Pirmame pociklyje:

- į D ateis MBR buvusi reikšmė 0001
- X bus nustatytas į 15_{10} (000F).
- Į sumatorių kairėje bus pasiųsta MBR reikšmė 0001 praėjusi pro atvirkštinį kodą (COM) tapusi FFFE, dešinėje bus pasiųsta C reikšmė (pradinė MBR likusi iš pirmos komandos, kurios nežinojome).

Antrame pociklyje:

- bus sudėta FFFE ir C registre buvusi pati pradinė prieš komandų vykdymą buvusi MBR reikšmė ir rezultatas išsaugotas A registre.

Atsakymas: 0000

Papildoma informacija

Tai yra įvairaus pobūdžio papildoma informacija, siekiant, kad konspektas būtų kiek galima aiškesnis. Taip pat, kad ruošimasis egzaminui būtų efektyvesnis ir sklandesnis.

Įvairios sąvokos, taisyklės ir pastabos

Akumuliatorius **–** žodinis registras AX, arba baito registras AL

Betarpiškas operandas **–** operandas konstanta, konkretus skaičius naudojamas kaip vienas iš komandos operandų. Pvz.: komandoje *mov ax, 12h* turime betarpišką operandą 12h. Betarpiškame operande kitaip nei atmintyje ar registre neįmanoma išsaugoti kokios nors reikšmės, pvz.: *mov 12h, ax* būtų neteisinga komanda.

Invertavimas (en. complement) – bitų reikšmių priešingomis pakeitimas (vienetiniai bitai keičiami nuliniais, o nuliniai vienetiniais), pvz.: invertavus baitą 0011 0110 turėtume 1100 1001

Plėtimo pagal ženklą taisyklė – pridedant poslinkį, esantį viename baite jį reikia suprasti kaip skaičių su ženklu, todėl pridedant jį prie efektyvaus adreso arba kokio nors žodinio registro reikšmės jį reikia plėsti pagal ženklą. Plačiau apie tai [čia]

Prefiksas/prefiksinė komanda – prieš komandos operacijos kodą einantis baitas (gali būt ne ne vienas), įtakojantis segmentų komandoje naudojimą (segmento keitimo prefiksai) arba kiek kartų reikės vykdyti komandą (pakartojimo prefiksai). Iš tiesų jei panaudoti 3 prefiksai prieš vieną komandą, bus įsiminti tik 2, bet nėra apibrėžta kurie.

Rezultato laukas – baito arba žodžio dydžio apribotas atminties laukas, kuriame saugoma reikšmė yra iš gana siauro riboto intervalo (pvz. vienas baitas gali įgyti reikšmę tik iš 256 t.y. 2^8 galimų).

DS:BX ir kiti tokie užrašai: nurodo absoliutų adresą atmintyje. Kairėje dvitaškio yra segmento registro reikšmė, o dešinėje efektyvus adresas. Šiuo atveju efektyvus adresas būtų BX reikšmė, o absolitaus adreso reikšmė *DS*10h+BX*

Steko viršūnės reikšmė – žodis, esantis steke su poslinkiu SP, jei paskutinis atliktas veiksmas buvo PUSH – tai yra vėliausiai steke padėta reikšmė.

Keletas pastebėjimų ruošiantis egzaminui

Siūlau atsižvelgti į šiuos punktus, tačiau **tai yra kiekvieno asmeninis reikalas**.
Dalis jų atrodo paprasti ir savaime suprantami, tačiau kaip sakoma „genialumas paprastume“.
Tikiuosi nepatingėsite paskaityti šiuos punktus, nes jie gali būti svarbūs.

1. **Pasitikrinkite** oficialiame egzaminų tvarkaraštyje, **kur ir kada vyks egzaminas**.
2. Atkreipti dėmesį, kad egzamino trukmė yra valanda ir gaunate 6 užduotis. Kiekviena užduotis vertinama vienodai (vienu balu). Taip pat negalėsite naudotis kalkuliatoriais.
3. Svarbu sugebėti perskaityti sąlygą, kiekvienas žodis joje svarbus.
4. Komandos pavadinimas dažniausiai duodamas, tačiau jei ir neduodama, dažniausiai kokia tai komanda nesunku suprasti iš duoto aprašymo tarkim „grįžimo iš procedūros komanda“ yra RET, „grįžimo iš pertraukimo procedūros“ IRET ir pan.
5. Jei sąlygoje pateiktas komandos formato aprašymas (tarkim valdymo perdavimo užduoties atveju pasakyta, kad valdymo perdavimas yra vidinis tiesioginis), tai nebūtina skirti pernelyg daug dėmesio mašininio kodo analizavimui ir nustatinėjimui, kuris komandos formatas tai yra, tačiau jei aprašymo nesuprantate ir geriau mokate formatą nustatyti analizuodami mašininį kodą, darykite taip, svarbiausia, kad uždavinį išspręstumėte teisingai.
6. Jei užduotyje prašoma registrų sumos (gali būti pasakyta ir „užrašykite *nurodyta kokie**registrų reikšmę**“ arba „apskaičiuoti registrų *nurodyta kokie* sumą“), tai apribojimų skaitmenų kiekiui nėra, tačiau prieš sudėdami kiekvieno registro reikšmę turite apskaičiuoti kiekvieną atskirai, t.y. atskiroms registrų reikšmėms apribojimai taikomi.
7. Jei užduotyje klausiama, kas bus įvykdžius nurodytą komandą (koks bus kitos vykdomos komandos absoliutus adresas (tai nurodo *CS:IP* pora), kokia registro reikšmė etc.), tai reikia suprasti, kad „atliekate debugerio žingsnelį“, įvykdote tik tą komandą, ir visiškai nesvarbu kas bus vykdoma po to, ar kas vykdyta prieš tai, nes tame žingsnyje esančią komandą jūs turite, registrų reikšmes taip pat. Į tai dažnai neatsižvelgiama tarkim, paklausus SP reikšmės įvykdžius komandą INT 21h, manoma, kad įvykdoma visa pertraukimo procedūra, nors realiai valdymas perduodamas tos procedūros adresu. Jei gaunate eilutinę komandą su pakartojimo prefiksu, tada turite įvykdyti ją visą, tie kartų kiek reikia, nebent nurodyta kitaip.
8. Riboti reikšmių laukai: baitai, žodžiai registrų reikšmės arba, priklausomai nuo registro, efektyvūs absoliutūs adresai turi būti surašyti su tiek skaitmenų, kiek turėtų maksimali reikšmė. Jei reikšmė nulinė tai tiek nulių ir parašote. Tarkim mažiausias absoliutus adresas yra ne tiesiog 0, o turėtumėte rašyti 00000.
9. MPL komandas rašykite griežtai pagal sintaksę, o ne „kaip trumpiau“. Kaip bus vertinamos teisingos, bet su bloga sintakse komandos, nežinau, bet mokant spręsti uždavinius MPLu reikėtų mokėti ir teisingą sintaksę.
10. Greičiausiai Jums bus duoti ***tik*** lapai su sąlygomis. Juodraščio lapu turėtumėte pasirūpinti patys. Jei to nepadarysite, užduotis teks spręsti tam pačiam lape, kuriam duotos sąlygos (toj pačioj ir kitoj pusėj)
11. Prie sąlygose pateiktų klausimų reikės surašyti gautus atsakymus. Ar teisingai užrašyti, ir ar išvis užrašyti atsakymai reikėtų peržiūrėti bent porą kartų. Tai turėtume pasitikrinti ir prieš pat atiduodami darbą.
12. Įmanomas uždavinio atvejis, kad atsakymo nebus įmanoma nustatyti.
13. Šešioliktainėje sistemoje 9+1 = A (o ne 10!)
14. Visi registrai ir atminties būsenos duotos atmintyje yra žinomi prieš pradedant komandos vykdymą, nebent išreikštinai nurodyta kitaip (komandos vykdymas susideda iš 3 esminių žingsnių: Komandos skaitymas iš atminties; Komandos atpažinimas ir komandų skaitliuko padidinimas; Komandos operacijos įvykdymas.